12-2005

Enhancing System-on-Chip Verification Using Embedded Test Structures

Wei Jiang
University of Tennessee - Knoxville

To the Graduate Council:

I am submitting herewith a thesis written by Wei Jiang entitled "Enhancing System-on-Chip Verification Using Embedded Test Structures." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Don Bouldin, Major Professor

We have read this thesis and recommend its acceptance:

Gregory Peterson, Syed Islam

Accepted for the Council:
Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

To the Graduate Council:

I am submitting herewith a thesis written by Wei Jiang entitled “Enhancing System-on-Chip Verification Using Embedded Test Structures.” I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirement for the degree of Master of Science, with a major in Electrical Engineering.

Don Bouldin
Major Professor

We have read this thesis
and recommend its acceptance:

Gregory Peterson

Syed Islam

Accepted for the Council:

Anne Mayhew

Vice Chancellor and
Dean of Graduate Studies

(Original signatures are on file with official student records.)

Enhancing System-on-Chip Verification Using Embedded Test Structures

A Thesis
Presented for the
Master of Science Degree
The University of Tennessee, Knoxville

Wei Jiang

December 2005

Dedication

This thesis is dedicated to my fiancée, Erika Fuentes, who always believed in me and supported me.

Acknowledgement

First, I would like to thank my thesis advisor, Dr. Don Bouldin for his guidance, advice and support throughout my research. Also, I would like to thank Tushti Marwah, for her teamwork in this project and her help with post-silicon tools. I am thankful to Rishi Srivastava for his discussion and previous work on the baseline design. Specially, I would like to express appreciation to DAFCA Inc. for giving us access to their software and their help in pre-silicon instrumentation.

Abstract

In this project, we designed and implemented a System-on-Chip platform with embedded test structures. The baseline platform consists of a Leon2 CPU, AMBA on-chip bus, and an Advanced Encryption Standard decryption module. Reconfigurable test logic blocks were embedded to form the test structure that can be used in post-silicon debug and verification.

The System-on-Chip platform was designed at the register transistor level and implemented in an 180nm CMOS process. Test logic instrumentation was done with DAFCA, Inc. (Design Automation for Flexible Chip Architecture) pre-silicon tools. The design was then synthesized using the Synopsys Design Compiler and placed and routed using Cadence SOC Encounter. Total transistor count is about 2 million, including 800K transistors for original platform and 700K for debugging module serving as on chip logic analyzer. Core size of the design is 3mm x 3mm and the system is working at 15MHz. Design verification was done with Mentor Graphics ModelSim and Cadence NCSim. Simulations were also used with the post-silicon environment to verify the functionality of the embedded test structure.

With a baseline platform ready and verified, designers can obtain high quality derivative designs quickly and easily. The visibility and the controllability of internal signals can greatly accelerate the testing and debug process, while the ability of post-silicon logic fixing can be used to verify design patch and enhance the reliability of the design. The whole design flow is cost effective in multi-million-transistor design because re-spins and delays in bringing a product to market may be avoided.

Table of Contents

List of Tables

List of Figures

Chapter 1 Introduction

1.1 Evolving of Integrated Circuits

The first integrated circuits (ICs) contained only a few transistors. In early Small Scale Integration (SSI) ICs, the number of transistors in the circuits was in the tens and contained only a few basic logic gates. The next step in the development of integrated circuits was taken in the later 60's, when devices contained hundreds of transistors on each chip, defined as Medium Scale Integration (MSI). They were attractive economically because more integrated chips allowed more complex systems to be produced using smaller boards. Driven by the same economic factors, further development led to Large Scale Integration (LSI), with tens of thousands of transistors in one chip. Now, in the era of Very Large Scale Integration (VLSI), with millions of transistors in a single chip, we have 64-bit microprocessors with on-chip cache memory and floating-point units (FPU). Today's processor, such as the Pentium 4 CPU from Intel, has 42 million transistors in a single chip.

The rapid growth of the semiconductor industry has followed Moore's Law [1], which was predicted by Gordon Moore. Although it has been rephrased and whether it is still holds true are under arguments, it's true that the size of transistors has been shrinking exponentially in the past 30 years, which resulted in the explosion of the transistor count in integrated circuits. Figure 1.1 illustrates that the number of transistors integrated into the CPU has been steadily increasing. This led to higher computing performance when measured as millions of instructions per second (MIPS). At the same time, the cost of integrated circuits has been decreasing as well. Silicon-based components became exponentially cheaper to produce [2, 3]. Today, integrated circuits are everywhere in our daily lives, from toys and musical greeting cards to supercomputers and space shuttles.

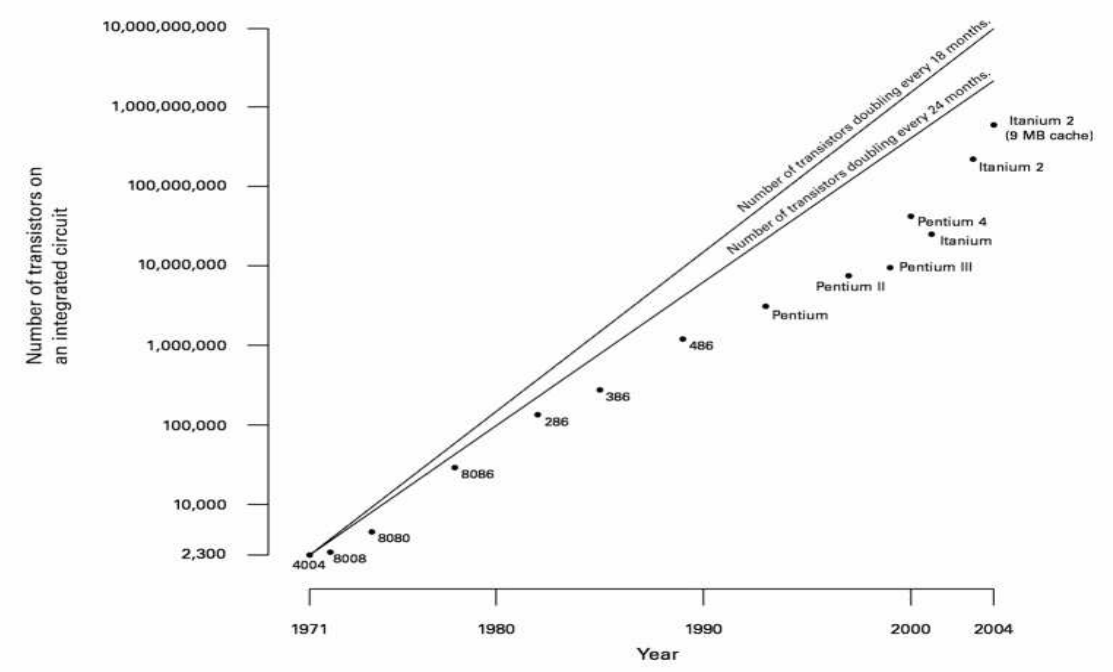


Figure 1.1 Moore's Law [3]

1.2 Design Technology

Design Technology is the most important factor in the realization of a microelectronics system. Its components include tools, library, manufacturing process and design methodologies. Today, the cost of design is the greatest threat to continuation of the ITRS semiconductor roadmap [4]. The manufacturing non-recurring engineering (NRE) costs are in the order of one million dollars, while the design NRE costs are in the order of tens of million dollars. The main reason is that design errors result in silicon re-spins which will multiply the manufacturing NRE costs.

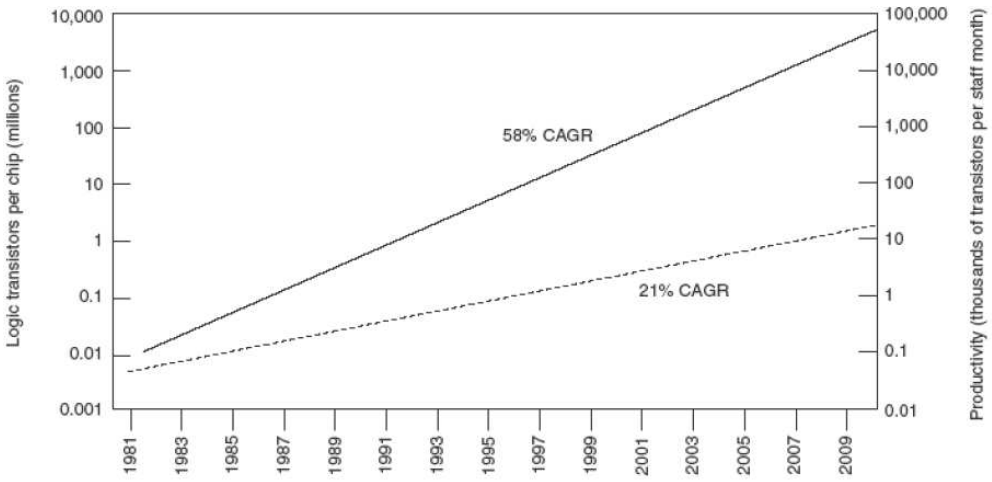
Currently the feature size is already in the deep sub-micron range (<90 nm). Designs have become extremely complex and more physical effects can no longer be ignored. Still, most of the investment in design technology is in process technology. As a result, the design ability has far lagged behind the capability of manufacturing [5]. There is a gap between design productivity and manufacturing

capability. In Figure 1.2 we can see that designers are not able to fully utilize the gate density afforded by a modern silicon process. The disparity is called the *design gap*.

Current design methodology has become the bottleneck of the design technology. As its consequence, we see that the design and test cost are growing exponentially relative to manufacturing cost. Verification engineers outnumber design engineers in a complex design development. There is a very urgent need calling for new tools and design methodologies.

1.3 Design Testing and Verification

While the manufacturing processes are improving, design errors and defects will continue to occur, especially when the process is at deep sub-micron since more physical errors are bound to happen. The existence of defects and errors imply the need for design testing and verification. Design verification refers to predictive analysis to ensure that the synthesized design, when manufactured, will perform the desired I/O function.



Source: Semiconductor Industry Association. *International Technology Roadmap for Semiconductors*: 1999 edition. Austin, Texas: International SEMATECH, 1999

Figure 1.2 Design Gap

A typical digital IC design flow is shown in Figure 1.3. Given user requirements, a designer has to go through specification, system level design, logic design, physical design and fabrication to obtain the final product. Design verification or test needs to be done at each step to make sure that the final product will meet user requirements.

Design verification and testing is becoming the most stressful part of the design flow. In current semiconductor industry, designers are spending 70% of their time on design verification [6]. The key challenges in design verification are:

- Increasing complexity of the design (approaching billion transistors)
- Rapidly increasing mask costs (approaching 10 million dollars)
- Time to market shrinking

In order to overcome these challenges, we need better testing algorithms and structures, better debug methods to locate the design/physical errors and more efficient ways to verify the fixes.

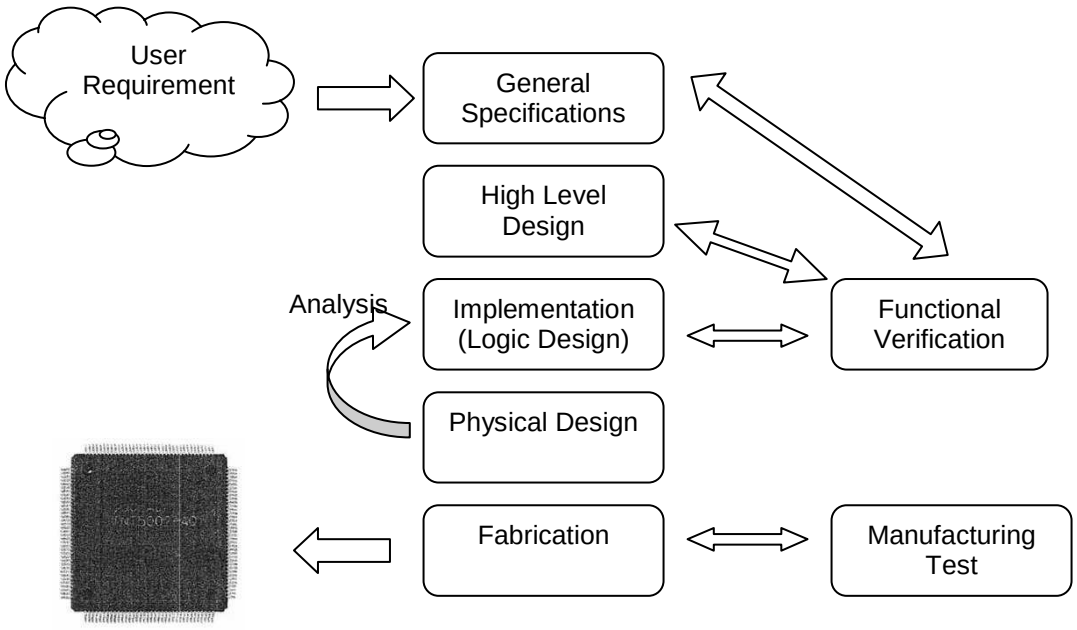


Figure 1.3 Typical Design Flow

Because of the reasons stated above, designers should have a test plan in mind during the design process. Currently, an increasing number of circuits are designed in accordance with Design-For-Testability (DFT) guidelines, which makes the design easier to be tested and debugged. The payoff is not only higher quality, but also shorter time-to-market [7]. The testability problems for digital circuits can be classified as controllability problems or observability problems (or both). Controllability is the measure of the difficulty with which an internal net can be driven to a particular logic state. Observability is the ability with which an internal signal's current logic state can be driven to an output where it can be measured. It should be noted that very often observability is a function of controllability, because if an internal net cannot be driven to a particular state, some node may be impossible to be observed from the output. Hence, the desired testing structure should deliver both of them for testing and debug.

1.4 Project Motivation

In order to reduce the risk of integrating a large system with the pressure of a fixed deadline and perform prototype debug/verification rapidly to reduce a product's time-to-market, new design approach and techniques need to be employed. Design methodology has evolved into System-on-Chip (SoC) design, which incorporates whole systems on a single chip. It uses intellectual property (IP) blocks as virtual components and greatly increases a designer's productivity. At the same time, the emergence of System-on-Chip design has brought a lot of challenges. Most of today's virtual components (i.e. IP cores) don't have well-defined contents and interfaces; they are often fuzzy and more like "patches in a quilt, which have to be carefully stitched together" [8]. Hence, integrating existing IP blocks to form a larger system is not a simple task. Moreover, with the rapid shrinking of the feature size, more physical errors are bound to occur due to timing, crosstalk, noise, temperature, and process variation. At the same time, the designers are losing visibility into the design as the size of the design

increases [9]. The internal pins are buried inside the chip and this makes it even harder to find errors in the design after the chip is fabricated. The whole debugging process requires multiple re-spins and delays the product's time-to-market by several months.

The motivation of this project is to use the new platform-based design methodology to perform system integration seamlessly, together with embedded reconfigurable logic that can accelerate the prototype debug and verification process. The goal is to design and implement a System-on-Chip platform that can be easily used to implement other derivative designs. At the same time, we are also exploring the new solution to the SoC debug problem by using embedded reconfigurable logic modules to provide observability and controllability to designers during prototype verification.

The implementation of the project involves IP core selection, system designing, interface standardization, debug strategy planning and integrating design tools from different vendors. The actual implementation of the design is targeting an 180nm CMOS process but it can be easily switched to other processes.

The baseline system was built with help of previous work of other students at University of Tennessee. I have modified the configuration of the SoC and rebuilt the baseline system. The system testing and debug plan was considered and DAFCA testing logic blocks were inserted. More simulations were done to verify the integrity of the baseline design and also the basic functionality of the embedded testing logic.

1.5 Thesis Outline

In this section we briefly introduced the challenges that current design technology is facing and the need for new design methodologies and tools. In Chapter Two we will discuss some background information such as the design trends and

challenges in SoC design. Chapter Three explains the Volunteer SoC platform developed at the University of Tennessee and its components. Chapter Four discusses about the SoC debug and verification challenges and possible solutions. Chapter Five presents the design implementation in detail and our results. Chapter Six concludes my work and presents our plan for future work.

Chapter 2 Background

With the rapid shrinking of process feature size, the transistor count has been exploded in the current IC design. These developments have shortened the product life cycle and made product time-to-market a critical issue. The pressure of developing a complex system by a fixed deadline pushed designers to employ the design methodology that is based on existing designs. SoC design has become the trend of the semiconductor industry since it has extensive design reuse. This chapter discusses about the advantages as well as the challenges of the SoC design methodology. Some solutions are presented including the use of a platform-based approach for rapid high quality development and embedding reconfigurable logic to enhance the flexibility of the design.

2.1 Design Reuse

A very important technique to exploit the capability of IC manufacturing is to reuse previous designs. Since the designer is given pre-verified components, most of the design doesn't have to be done from scratch. This greatly shortens the design cycle and increases the quality of the design.

In order to assemble a system with reusable blocks, first each component must be *known good*: they should be pre-tested and documented in detail for fast integration. This approach is the best when the components are designed with the purpose of reuse in mind. In a system based on reused blocks, unproven components and interconnect are riskier and more time-consuming to verify. Hence the focus of design of such a system is mostly in components verification and interface designing [5].

Design reuse can have a significant impact in the development cycle of a product. Since each sub-design is developed for reuse, it will take more time and effort for interface standardization, thorough testing and detailed documentation.

From Figure 2.1 we can see that when there is no planned design reuse, the total development time is proportional to the number of sub-designs, given that each module has the same complexity. For development with planned design reuse, the designer will spend more time in developing a reusable sub-design in terms of documentation, interface design, etc. Still all the extra effort and time will pay off when we integrate all the blocks into a large system because designing with IP is much faster than from scratch. The final design will have better quality and require less time to be ready for market entry.

2.2 System-on-Chip Design

System-on-chip is a new product class in design methodology. With today's silicon technology, designers can put all the functionalities of a system in a single chip package. Such high level integration has lots of advantages, such as lower latency and higher communication bandwidth, fewer discrete components and less area, and is hence more cost effective. One of the most important features of SoC design is that it can integrate different technologies and other design elements such as microprocessor, embedded memory, analog and mixed signal circuits as well as reconfigurable logic [4]. It has high integration complexity and usually for high volume production to reduce the design costs, as in traditional Application Specified Integrated Circuits (ASICs). The key point of SoC design is to maximize the reuse of existing blocks or IP cores and minimize the modules that are newly created.

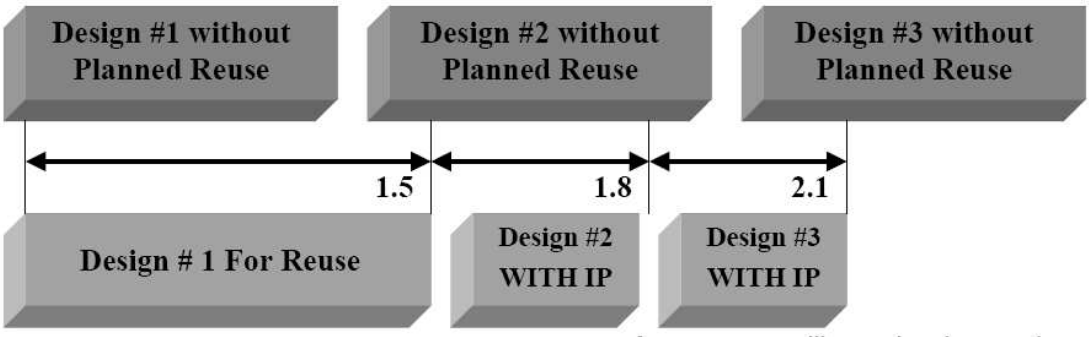


Figure 2.1 Design Reuse [5]

As in any design, cost is the most important consideration and the driver for the evolving of SoC design. It is characterized by heavy reuse of the intellectual property blocks to improve design productivity; hence it provides low cost and high integration. The cost consideration requires SoC design to use a low power process, low cost packaging, easy system validation and fast design turn around time, which requires a new design methodology.

Today, SoCs can have as many as several tens of million gates, multiple IP cores, and complex on-chip buses and protocols. The integration of all the components into a system and the verification of such a big design has become a very challenging job [10]. These are described in the next section.

2.3 Challenges in System-on-Chip Design

With the emergence of SoC, new challenges have been presented, such as integration of complex blocks, mixing of software and hardware, and design verification and debugging. A few years ago the majority of the silicon re-spins were due to simple functional design errors, and designers could make simple assumptions to predict and compensate for the impact of physical effects such as signal integrity and crosstalk. Today with nanometer processes, these simple assumptions cannot model the physical effects in the deep sub-micron range. SoC designers are finding more and more errors are due to physical effects that are not reflected in the simple models being used. As a result, a more reliable design methodology and verification techniques are needed [9].

The designers are facing explosion in design complexity because of the advancing of today's manufacturing technologies. It can be divided into two categories: system complexity due to the increasing size of the transistor count and silicon complexity due to physical effects in the deep sub-micron range.

2.3.1 Silicon Complexity

Silicon complexity refers to the impact of process scaling and the introduction of new materials or devices. Many phenomena that were ignorable previously now have great impact on design correctness, such as:

- Non-ideal scaling of device parasitic and supply threshold voltages
- Coupled high frequency devices and interconnections
- Manufacturing variability
- Scaling of global interconnect performance relative to device performance
- Decreased reliability
- Complexity of manufacturing handoff
- Process variability

2.3.2 System Complexity

System complexity refers to the exponential increase in transistor count enabled by smaller feature sizes and required by consumers demand for increased functionality, lower cost and shorter time-to-market. Design specification and validation have become extremely challenging, particularly with respect to a complex operating context. Tradeoffs must be made between quality and cost:

- Design reuse
- Verification and test
- Cost-driven design optimization
- Embedded software designed
- Reliable implementation platforms

2.4 Platform Based Design

An extension of the core-based design methodology is Platform-Based Design (PBD). It creates highly reusable groups of cores to form a complete hardware platform, further simplifying the SoC design process. With highly programmable platforms that include one or more programmable processors and reconfigurable logic, derivative designs may be obtained easily by adding and dropping a few IP

cores. Platform customization for a particular SoC derivative then becomes a constrained form of design space exploration, because the basic communication protocol and processor choice are fixed. The design team is restricted to choosing certain customization parameters and the virtual components from the library [4].

Platform-based design is becoming the method of choice for designing SoCs for embedded systems [3]. It has extensive planned design reuse, which enables designers to create a succession of derivative designs. In this approach, the main focus areas for the designer are interface standardization, virtual system design, and designing the system architecture and interface between the blocks. The basic idea behind this is to re-use significant portions of previous designs to reduce the time-to-market, which generally results in greater revenue for the product. Under this concept, the first goal is to develop a complete SoC that is central to a product line. Usually there is a processor, a real-time operating system, peripheral IP blocks, some memory and a bus structure. Once the baseline platform is fully functional, a derivative design in which only a few virtual components are added or dropped can be accomplished easily [4].

2.5 Reconfigurable Logic

Reconfigurable devices such as Field Programmable Arrays (FPGAs) and ASICs are two main design approaches in the current IC development. Although ASIC design has it advantage in superior performance, the NRE charges are very expensive and the longer development cycle make it unsuitable for some applications. On the other hand, compared to traditional ASICs, an FPGA has a great advantage in flexibility since it can be reconfigured after the silicon is fabricated and it’s much easier to debug and prototype. A new trend in current IC development is to embed an FPGA-like fabric in the ASIC design to take advantage of both approaches. Although the concept of embedding reconfigurable logic into traditional ASIC has been around for awhile, the reason

that it has not made a huge impact in the IC design is a lack of clear investment return and software support. Since all the reconfigurable logic will have large overhead in performance (wiring delay) and area (transistor counts), the design engineers need to justify the decision of choosing hardware to achieve reconfigurability rather than through embedded software.

Up until now, most of embedded reconfigurable logic is used as glue logic or as a part of design with flexibility to implement future functions. In the near future, one of the new fields that will make reconfigurable logic attractive is going to be in debugging and system verification. As we know, with millions of transistors in the design, SoC debugging and verification is extremely difficult. During a product development, multiple re-spins are not uncommon. The masking cost of each re-spin is as much as several million dollars, while the delay of product's time-to-market also costs profits of the product. A new approach of silicon debugging is to embed reconfigurable logic in the system to help debugging and verifying design. The extra embedded reconfigurable logic will help designers find the design and fabrication error and give the designer much more flexibility to debug and verify the fabricated design. Hence it will reduce the number of re-spins and product's time-to-market. Overall it is much more cost effective compared to the traditional SoC debugging method.

2.6 System-on-Chip Debug

System-on-Chip debug and verification has become one of the hardest steps in a product development. With the transistor counts in the design increasing exponentially and feature size going to the nanometer range, more physical and design errors are going to occur. System verification and debug for such a huge design can take months of lab time and several re-spins of prototyping [11].

Although simulation is still essential for design verification, it's simply not possible to simulate every scenario that a design will have in a target system. Moreover,

verification techniques cannot cope with the exponential increase in the complexity that results from the integration of SoC blocks. Inter block communication requires a perfect interpretation of multiple hardware/software specifications, each of which can be several hundred of pages long. If simulation coverage is limited and if the specifications are prone to error and misunderstanding, it's no surprise that nearly two-thirds of all designs require one or more re-spins [9].

2.6.1 Board Level Vs Chip Level Debug

Software flexibility and signal observability are essential for board level system integration and validation, and enable the effective exercise of system functionality. The ability to observe and control individual signals with tools such as logic analyzers and pattern generators is indispensable for localizing and diagnosing errors. Additionally, special diagnostic codes can be loaded into board level components, thus expediting the validation and debug of critical interfaces in a process commonly called software debug.

Unfortunately, in post-silicon SoCs, while software control is still available, observability and controllability of hardware is practically eliminated. At the board level, observation requires a probe connecting to the metal trace of the signal. But at the chip level, only the largest semiconductor companies can afford such kind of probing, since the internal signals are often buried under several layers of metal and silicon. For regular designers, the lack of observability and controllability can severely hinder the SoC validation process. As a result, often bugs exist in silicon and lead to re-spins and delay of product market entry.

2.6.2 Debug Method

There are several debug methods that are currently employed in chip level debugging [12]. Each of them has its own advantages and disadvantages.

- Full Scan/Partial Scan/Single Step

Full scan or partial scan is a debugging technique that uses the scan registers to provide observation and control over the circuit under test. A circuit can be tested at-speed but has to be stopped to retrieve data from the scan chain. While state information on all scan registers can be retrieved, state information of only a few clock cycles is typically available. So this debug method requires an iterative approach of starting, stopping, and restarting the circuit under test in order to isolate a problem. This is often the most effective method to isolate problems. Unfortunately many circuits cannot be stopped and restarted, and even fewer circuits can be single stepped.

- At-speed observation

While a scan chain provides a wide view of signal states, many chips are inserted with debug muxs and tracer memory to provide a means to see a narrow but deep view of the selected signals. This is similar to using a logic analyzer and a user can build triggers and capture data as a means to isolate a particular problem. While this method can be effective, it's often iterative, as the user often must create many triggers and many signal states to reduce the scope of the problem continuously before it's fully isolated.

- Assertion-based debug

Assertion-based debugging is a relatively new debug technique that uses assertions to monitor the behavior of circuits. In some cases, the assertion is hard-coded into the design at the register transfer level (RTL) while in other cases it can be implemented as reconfigurable logic and can be configured after the chip is fabricated. While some assertions can pinpoint a problem, most only limit the scope of a problem. Often additional methods are required to isolate a problem fully.

- At-speed Control

At-speed control refers to debug techniques that allow a user to dynamically modify the behavior of a circuit running at-speed. This is simply the control over a configuration register changed on the fly. In other cases, it may involve the dynamic reconfiguration of a programmable circuit. At-speed control does not include the ability to change scan registers and configuration registers. Such control is provided with full scan, while at-speed control just implies the ability to modify the behavior of a circuit without stopping the circuit.

2.7 Embedding Test Logic in an SoC

During post-silicon, the visibility of the design inside the chip usually requires using a silicon probe, which is extremely expensive and inefficient. As a result, only the biggest semiconductor companies can directly access internal signals at the chip level. Most of the designers use the alterative approach by embedding test logic inside the design such as scan chain, or user-inserted ad-hoc observation structures. The problem is that designers can only access the scan chains with constraint test patterns and usually single-step, while user-inserted observation structures are inflexible and very difficult to be placed at the right place in design step.

Currently there are several silicon debug tools that embed testing logic in a design for an FPGA or ASIC. For example, many FPGA vendors provide an embedded logic analyzer and debug tools for design verification, such as Xilinx's Chipscope™ and Synplicity's Identify™. Other product such as FS2 Bus Navigator™ can monitor signal activity for debugging a complex bus system in SoC [13, 14, and 15]. Although these tools can greatly help designers locate an error in the design, their abilities are somewhat limited. Many tools are vendor-specific, while others could only apply for a particular system structure. More importantly, current debug tools can only provide design observability to designers, while controllability – another important requirement for design verification is not addressed.

New tools have already shown this new trend in SoC debugging. Software and hardware combined solutions are already available to embed testing logic in SoC designs to provide not only visibility of the chip, but also to control internal signals that efficiently verify or debug the design. DAFCA Inc. (Design Automation for Flexible Chip Architecture) provides pre-silicon and post-silicon tools for embedding reconfigurable cores in SoC design. In our project we are exploring the potential of using embedded test logic for rapid design debug and verification, to avoid re-spin cost and reduce product time-to-market.

Chapter 3 Volunteer SoC Platform

The Volunteer SoC platform was developed as part of a graduate course in the Electrical and Computer Engineering Department of the University of Tennessee over the last two years. The IP cores library were developed and verified by different student teams. The design can serve as an industrial strength design for students to exercise different CAD tools and learn about optimization techniques at the logic and physical levels. The baseline SoC platform, which used only open cores, can be obtained by anyone at no charge so that others may contribute to its enhancement or develop derivative designs.

In this section, we discuss the basic components of our SoC baseline platform in detail, as well as some technical consideration in integration process of the system.

3.1 Overview

Our SoC baseline design has a Leon2 SPARC-V8 CPU with caches, attached to an AMBA on chip bus. There are two bus architectures presented in the chip, AHB for high-speed data transfer and APB for on chip peripherals. Two buses are connected through a bridge. A memory controller is used to communicate with external memories. Two IP blocks are attached to both buses as user modules. The first IP block is an encryption module that performs 128-bit decryption according to the Advanced Encryption Standard (AES). The second block is a reconfigurable block array that is inserted with DAFCA pre-silicon tools. The details of the IP blocks will be discussed later in this chapter. The design is mostly open core-based since the Leon2 is an open source CPU and user blocks such as AES can be obtained free of charge from an open source online IP repository [16]. The SoC is designed at the register transfer level and can be fully synthesized to different processes. In our implementation we used a 180 nm CMOS process. The block diagram of SoC is shown in Figure 3.1.

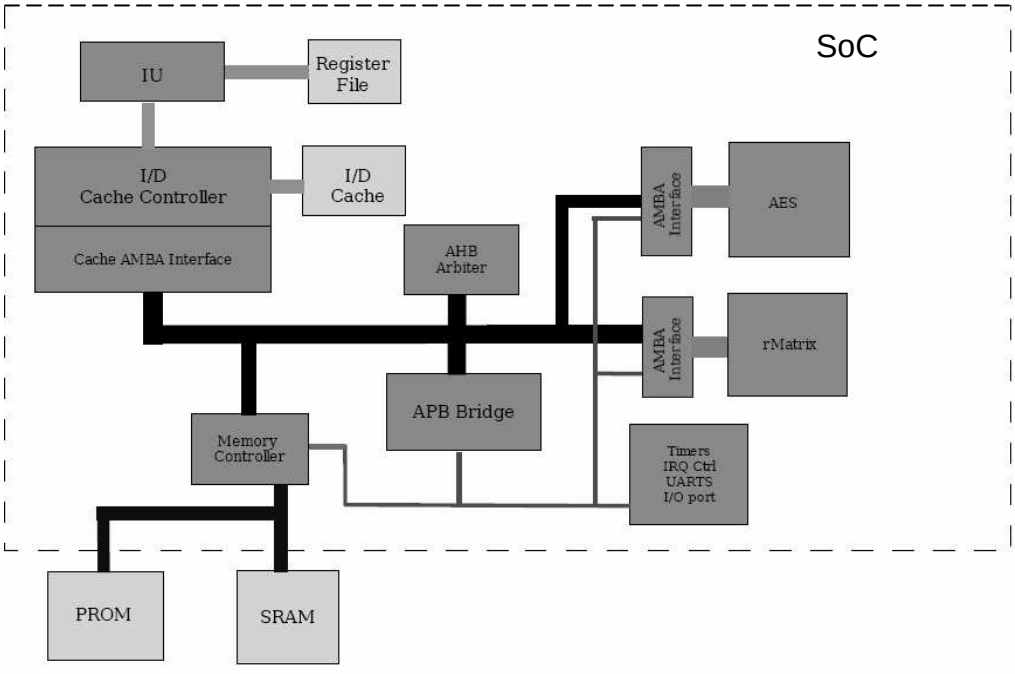


Figure 3.1 SoC Baseline Platform Block Diagram

Two user IP blocks are attached to both the APB bus and the AHB bus. All the IP blocks are defined as additional masters on the AHB bus and slaves on the APB bus. In our design, the AHB bus is used for data transfer between memory and IP blocks and the APB bus is used for control signals. The AMBA interface is provided in order to connect different user IP blocks to the AMBA bus. Although different user IP blocks have different I/Os, we have pre-defined a general scheme for control signals so the AMBA interface can be easily modified to attach different IP blocks from the library.

3.2 LEON CPU

The Leon2 CPU is a 32-bit SPARC-V8 (IEEE-1754 standard) CPU that was developed by the European Space Agency [17]. The source code is written in VHDL and can be obtained online at no charge. The processor is highly

configurable and particularly suitable for SoC designs. The VHDL model of LEON is fully synthesizable with most synthesis tools and can be implemented in both ASICs and FPGAs. A block diagram of LEON can be seen in Figure 3.2. Some of the key features of Leon2 are listed below:

- Separate Instruction and data cache
- Hardware Multiplier and Divider
- Interrupt controller
- Debug Support Unit with trace buffer
- Two 24-bit timers
- Two UARTS
- 16 bit I/O port and a flexible memory controller
- APB for on-chip peripheral on chip register access
- AHB for high speed data transfers

Leon2 is available at the source code level, which is essential for an open core based platform. Having access to the CPU source code, we are able to modify some particular components to perform the system integration, such as the bus arbiter and bridge. At the same time, having internal visibility of components will greatly help us to understand many system design issues.

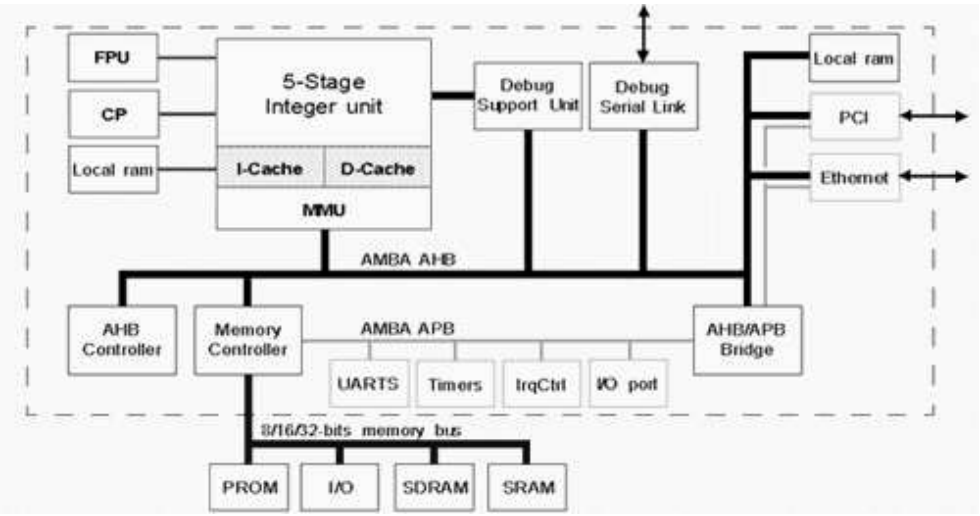


Figure 3.2 Leon2 Block Diagram [17]

Another important reason that we chose the Leon2 is that it is highly reconfigurable. Being a platform design, our SoC platform should have the flexibility to configure the microprocessor to suite a particular design since different projects have their particular needs. For example, in our baseline design, the Leon CPU is configured to minimum size in order to save silicon space. 1K of instruction cache and 1K of data cache are used; most of the optional components are disabled such as hardware divider/multiplier, on-chip debug unit etc.

3.3 AMBA On-Chip Bus

There are several interconnect specifications available for implementing a SoC on-chip bus, including IBM CoreConnect, ARM Advanced Microcontroller Bus architecture (AMBA) and Wishbone from Silicore Corp., etc. All of them address the same basic goal: connecting IP cores. They have similar bus topology and all provide basic handshaking and variable data bus sizes. While IBM CoreConnect is a complete and versatile solution and truly high performance, it is relatively more complicated and offers many features that will be unused in simple embedded applications [18]. Wishbone is a good simple specification for a peripheral bus but it suffers from its poor timing specification and lack of pipelining, which make it problematic in a high performance system [19]. AMBA is also an open specification that defines an on-chip communication standard for designing high performance embedded microcontrollers and relatively easy to implement [20]. There are three distinct buses defined within the AMBA specification:

- Advanced High performance bus (AHB)
- Advanced System Bus (ASB)
- Advanced Peripheral Bus (APB)

Particularly, AHB is for high performance, high clock frequency system modules and works as the high performance system backbone bus. It supports the

efficient connection of processor, on-chip/off-chip memory interface with low power peripheral macro-cell functions. It is also specified for ease of use in an efficient design flow using synthesis and automatic test techniques. ASB is an alternative system bus suitable for where the high performance features of AHB are not required. APB is optimized for minimal power consumption and reduced interface complexity to support peripheral functions. It is designed for low power peripherals and can be used in conjunction with an AHB/ASB system bus.

In our design AHB and APB are used as the on-chip bus architecture for our platform. The APB bus is used to access on-chip registers in the peripheral functions while the AHB bus is used for high-speed data transfer. The full AHB/APB standard is implemented in the Leon CPU and AHB/APB controllers can be customized through the TARGET package.

The processor in Leon2 is connected to the AHB bus through the instruction and data cache controllers. Access conflicts between the two cache controllers are already resolved locally and only one AHB master interface is connected to the AHB bus.

As for user IP cores, there are several ways of attaching user IP blocks on the AMBA bus: They can be attached to AHB and/or APB, as a master or as a slave. In our design we decided to attach them to both the AHB and APB buses in order to separate control signals and data transfer. The user IP blocks are defined as additional masters on the AHB bus and slaves on the APB bus. As a master on the bus, it has the ability to initialize a data transfer with bus slaves without waiting for the CPU, which is essential for a high performance system.

The AHB bus master interface is shown in Figure 3.3. It drives the data and address bus together with control signals to indicate the type of the data transfer (Read/Write, Sequential/Non-sequential, Protected/Unprotected etc). Before data

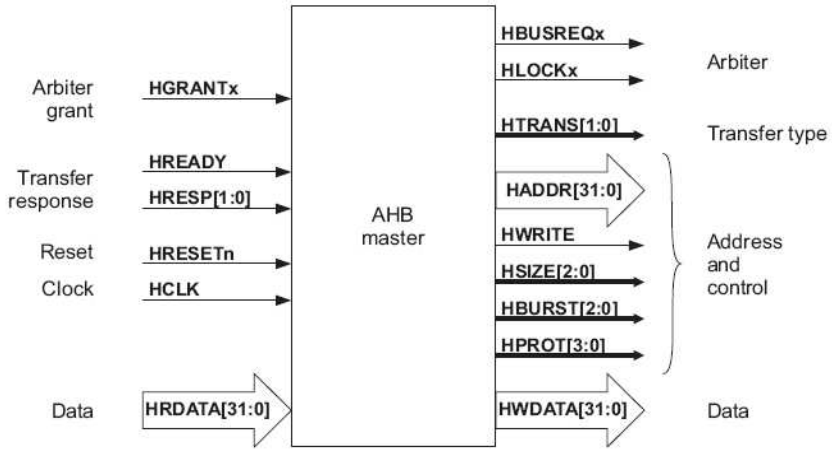


Figure 3.3 AHB Master Interface

transfer occurs, the bus has to be granted by the arbiter. The master will first request the bus from the arbiter. Once the bus is granted, the master can initialize a data transfer. Re-arbitration is done after each transfer unless it is a burst or locked transfer.

The user IP block also serves as an APB slave, whose interface is shown in Figure 3.4. As we discussed before, the APB has a simpler protocol since it's a peripheral bus. Being a slave, the IP block will wait for the selection signal from the bus master and read in the data/address bus, with control signals indicating the type of the data transfer. According to the command, it will perform a particular function and drive the read data bus on the APB.

3.4 AES Module

In our baseline design we are using an Advanced Encryption Standard (AES) module as user IP block. AES is a block-cipher/decipher with block size of 128 bits. Keys for the cipher come in one of three lengths: 128, 192, or 256 bits. In our design, for the purpose of achieving smaller design size, only a 128-bit key is supported. The particular AES module is a part of a cryptographic project developed at the University of Tennessee, while an open source version can be

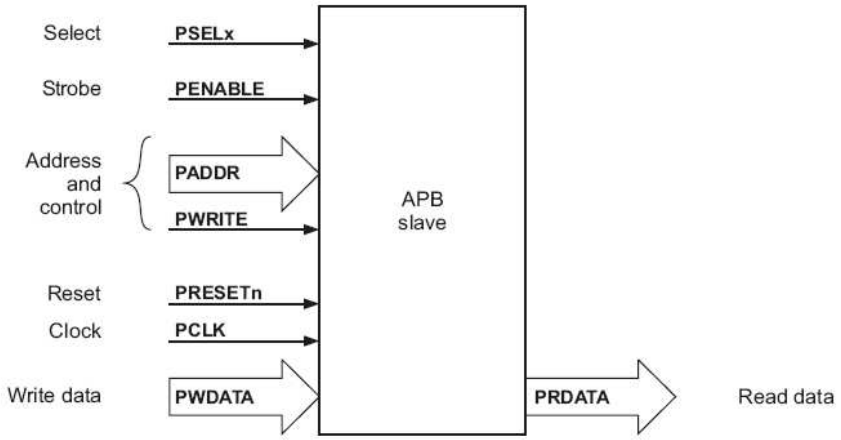


Figure 3.4 APB Slave Interface

obtained from opencores.org. The functionality of the block has been verified by simulation and tested in a Xilinx Virtex 1000E FPGA.

3.5 System Memory Address Mapping

The address space of Leon2 is 32 bits, which means that a maximum of 4 GB memory is supported. The address mapping for different address ranges is shown in Table 3.1.

The AHB bus is used to connect the cache controllers and memory controllers. The CPU is the only master on the AHB bus, while the memory controller and APB bridge are two slaves on the bus.

Note that the address space for the APB Bridge is from 0x80000000-0x8FFFFFFF. Since we attached the user IP block on the APB bus, we have to assign part of this memory space to the user IP blocks. This is done by modifying the source code of the APB Bridge. (apbmst.vhd)

Table 3.1 Leon2 Memory Address Space

Address Range	*Size*	*Mapping*	*Modules*
0x00000000-0x1FFFFFFF	512 M	PROM	Memory Controller
0x20000000-0x3FFFFFFF	512 M	Memory Bus I/O	Memory Controller
0x40000000-0x7FFFFFFF	1 G	SRAM/SDRAM	Memory Controller
0x80000000-0x8FFFFFFF	256 M	On-chip Registers	APB bridge
0x90000000-0x9FFFFFFF 0xB0000000-0xB001FFFF	256 M 128 K	Debug Support Unit Ethernet MAC Registers	DSU Ethernet

3.6 Artisan Block RAM

The cache system and the register file are implemented by using technology dependent RAM cells. In our design they are directly instantiated from the ARTISAN TSMC18 Library. The integer unit (IU) register file has one 32-bit write port and two 32-bit read ports. We are using the default number of register windows, which requires 136 registers. This is implemented as two 8x136 dual-port RAMs, each one with one read port and one write port. The register file read/write timing is shown in the Figure 3.5.

The timing specification for read/write access of the synchronized RAM generated by Artisan is shown in Figure 3.6. It should be noted that for Leon2, the register file must provide the read data at the end of same cycle as the read address is presented, which is different with the timing specification for Artisan SRAM. In order to use the block RAMs generated by the Artisan RAM generator, we used a memory wrapper in which the synchronous register file is clocked on the inverted clock to meet the timing requirement. Artisan synchronous single port RAM cells are also used for both tag and data in the instruction/data cache.

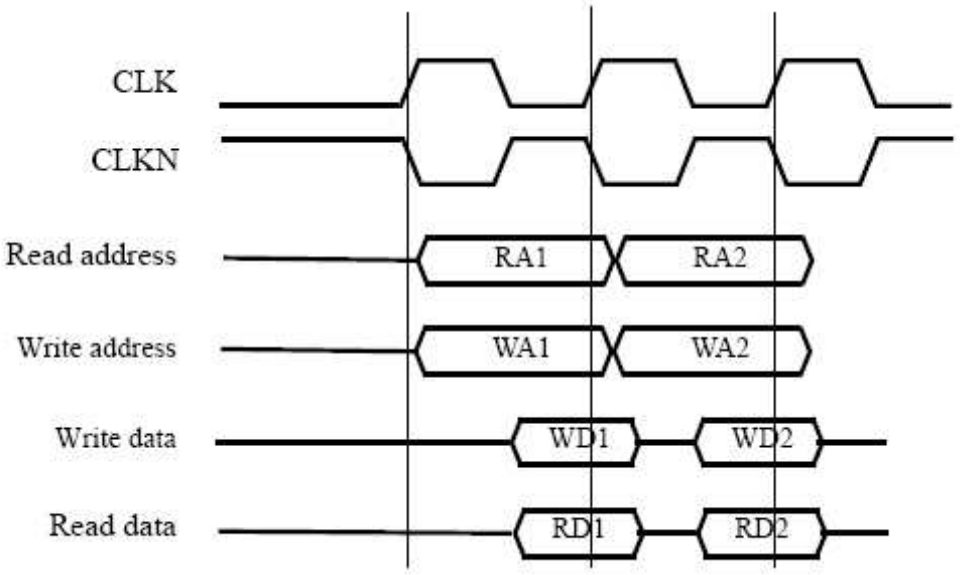


Figure 3.5 Leon Register File Timing

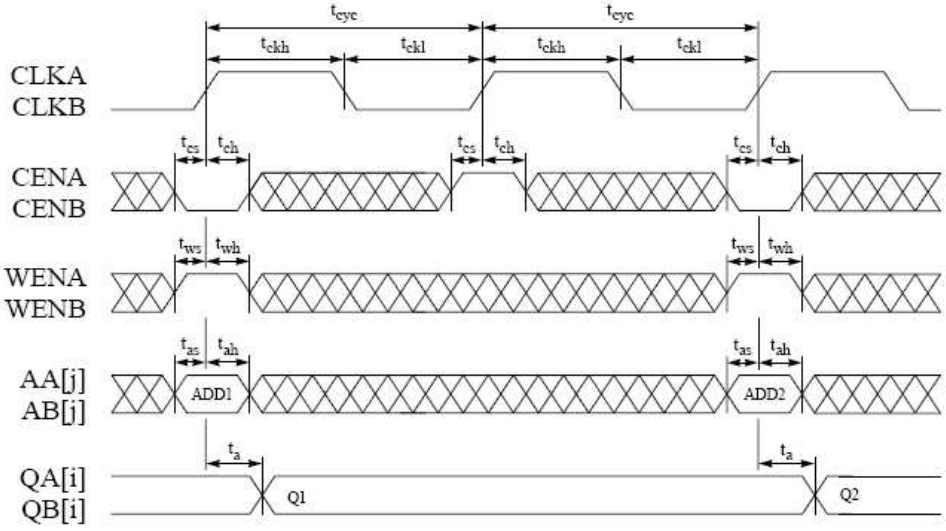


Figure 3.6 Artisan RAM Write Timing

The cache set size is set to 1KB and each line is set to have 8 words. This setting is chosen to achieve minimal size for both tag RAMs (32x30) and data RAMs (256x32).

3.7 Cross-compiler and Embedded Software

Since Leon2 is SPARC-V8 compliant, compilers and kernels for SPARC-V8 can be used with Leon. For simple embedded software development, Leon uses a real-time kernel RTEMS and a free C/C++ cross compiler LECCS. Leon/ERC32 Cross Compilation System (LECCS) is a GNU based cross compilation system for Leon processors [21]. It allows cross compilation of C applications for LEON.

Real-Timing Operating System for Multiprocessor System (RTEMS) is a real-time operating system designed for embedded systems [22]. It is free open source system designed to be standard-compliant and has basic kernel features:

- Multitasking capabilities
- Homogeneous and heterogeneous multiprocessor systems
- Event-driven, priority-based pre-emptive scheduling
- Optional rate-monotonic scheduling
- Priority inheritance
- Responsive interrupt management
- Dynamic memory allocation
- High level of user configurability
- Portable to many target environments

Chapter 4 System-on-Chip Debug and Verification

Design debug and verification has become one of the biggest challenges in the System-on-Chip development. While a SoC design can have as many as 20 millions gates in a single chip, its debug and verification become one of the most stressful steps in a product development cycle. Normally SoC debug requires one or more re-spins and months of lab time to fix all the errors in prototype silicon. In this section we discuss the DAFCA debug approach that is being used in our SoC design. The available tools and library blocks provided by DAFCA are also discussed in detail.

4.1 Hardware Verification Vs Simulation

Simulation is a necessary step in the ASIC/FPGA design flow. In order to verify the design has desired I/O behavior, simulation should be done at each step of the design flow (system level, logic level, physical level). There are many types of simulation at different levels: Circuit Level, Switch level, Gate level, RT level, behavior level, HW/SW co-simulation, etc.

One of the biggest problems of simulation is the tradeoff between speed and accuracy. Higher abstract level simulation is always faster than the lower level. But lower level simulation is more accurate and more likely to represent the real behavior of final product. Although up until now, simulation is still the most dominate method for design verification, it has many limitations. First, simulation is always based on theoretical models so very often it cannot model the real system timing and environment. Second, with the complexity of design increasing exponentially, testbench development has become very difficult and time consuming. It's simply impossible to simulate every scenario and to test all the rare cases. Moreover, as we mentioned above, lower level simulation is extremely slow and with the complexity of today's design, it's almost impossible to simulate the whole system at the lower level.

Another way of doing design verification is using prototyping. Prototyping is defined as building a test system that (hopefully) will act like the real system. In IC development, prototyping usually refers to fabricating a test chip and verifying the system function in hardware. Prototyping is usually expensive because of mask cost but in general is quite accurate. Hardware verification can be used as a supplement to the software simulation. It is much faster compared to software simulation. For an embedded system, this is the first chance to validate the operation of hardware and software together. It can expose the bugs that were missed or not addressed in simulation, as well as many potential physical problems during the manufacturing process.

4.2 DAFCA Design Flow

DAFCA, Inc. provides a System-on-Chip debug solution, which can be integrated in the regular SoC design flow. Its approach is illustrated in Figure 4.1. The instrumentation of the reconfigurable logic blocks is done at the register transfer level and technology-independent. The reconfigurable instrumentation can enable the designer to isolate and diagnose the bug. At this point the designer can use simulation to verify that the instrumentation of the DAFCA blocks does not affect the functionality of the original design. Then we can go back to the normal design flow with logic synthesis, place and route and fabricate the design with embedded reconfigurable logic. At post-silicon, with DAFCA instrumentation in the chip, designers can perform in-silicon debugging by configuring the instrumentation. Designers can set traps and triggers and the wrappers can be programmed to do assertion and logic modification. The user can also run regular test cases and record the internal states in the tracer memory, which can be pulled out later to be viewed in a waveform viewer. Since the reconfigurable instruments give designers the observability and controllability inside the chip, rapid silicon debug and verification is made possible.

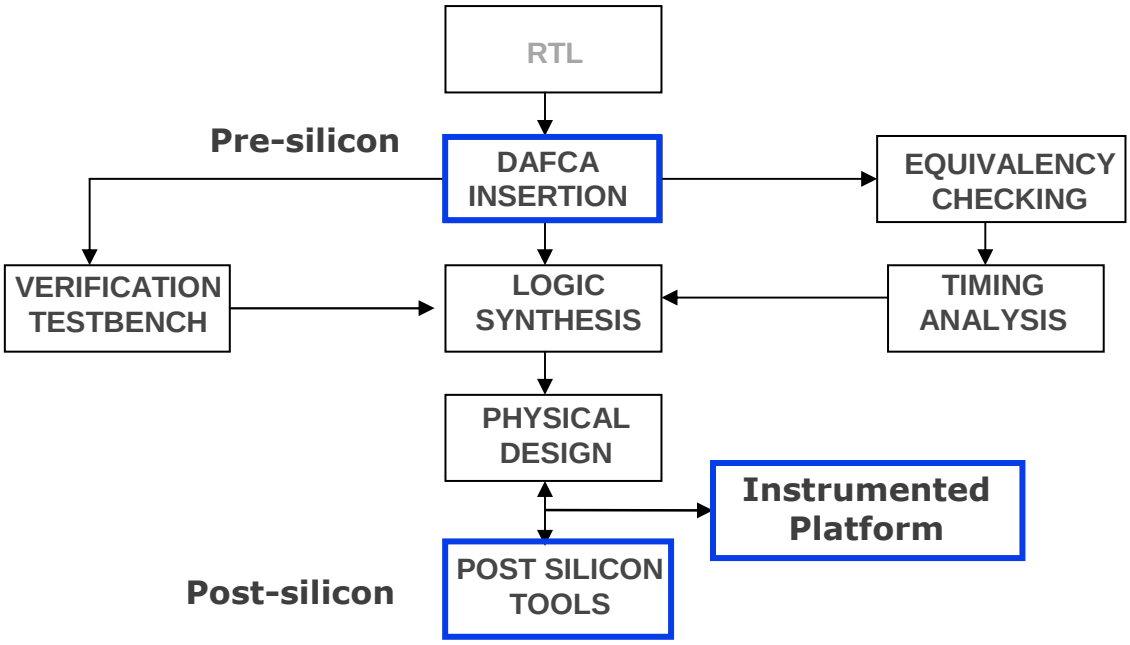


Figure 4.1 DAFCA Design Flow

4.3 ReDI ™ Instrument Library

DAFCA, Inc. provides a set of instrument blocks that can be inserted by designers, including the customizable instruments that designers generate with DAFCA tools, and also the library instruments with dedicated functions. All the instruments are inserted into a design as RTL and fully synthesizable. The brief functions of some instruments are shown in the Table 4.1. Among these blocks, rWrap, r1500, rMatrix are signal/port wrappers that can be used to give the designer the ability to control and observe a particular signal or port, while Primary Controller (PCON) and Serial Access Node (SAN) are blocks that are dedicated for communication between off-chip tools and on-chip instruments.

The properties of different instruments are compared in Table 4.2. As we mentioned before, r1500 is a wrapper that is compliant with the IEEE 1500 standard. It has the ability to observe an internal signal as well as the ability to drive it to a particular state. It cannot be programmed as in rWrap to modify user

Table 4.1 ReDI Instrument Library

ReDI Library Block	Block Functionality
Primary Controller (PCON)	Instrument that provides the on-chip interface between JTAG TAP controller and debug infrastructure
Serial Access Node (SAN)	Instrument that provides the on-chip interface between the PCON and the debug instruments inserted into user logic
rWrap	A one-dimensional reconfigurable instrument used for wrapping ports and signals
r1500	Customizable wrapper instrument compliant with IEEE std 1500
rMatrix	A two-dimensional reconfigurable instrument used for wrapping ports and signals
rMonitor	Instrument in the Debug Module that contains breakpoint and timeout counters, as well as reconfigurable instrument that can implement complex at speed assertions or triggers
CMUX	A highly configurable multiplexer used for tapping ports and signals

Table 4.2 Instrument Capability and Attribute

		Instrument Capability			Instrument Attributes			
		Observe User Logic	Control User logic	Modify User logic	Programmable	Mux Delay	Load	Size (gates)
ReDI™ Instruments	rMUX	✓			✓		✓	10 - 100 / signal
	r1500	✓	✓			✓		20 - 100 / signal
	rWRAP	✓	✓	✓	✓	✓		100 - 500 / signal
	rMATRIX[1]	✓	✓	✓	✓	✓		500 - 2000 / signal
	rMonitor[2]	✓	✓		✓			198k / chip

logic and it cannot perform at-speed debug. rMatrix is similar to rWrap in functionality, but is more complex and can perform more complicated logic since it's a two dimensional wrapper. At the same time it takes much more space in silicon.

rMonitor is the on-chip debug module that works like a logic analyzer. It is designed with a memory tracer to store the internal state of the system. Although the size of the rMonitor is about 200K gates per chip, it will pay off to have an on-chip logic analyzer for a multi-million gates design.

It should be noted that the r1500, rWRAP and rMatrix introduce an extra mux delay to the design, while rMux does not. This is because of the difference between the nature of tapping and wrapping a signal. More details will be discussed in the next section.

4.4 ReDI™ Insertion

As we can see from the previous section, implementing extra embedded debug logic will increase the size of the design. The choice of which ReDI blocks are going to be inserted into the design depends on a particular design and available silicon space.

When we are tapping a signal, we are referring to inserting an instrument to taps off an internal signal or port. With tapping, there is no new element that is inserted into the signal path explicitly, although a larger driver might be needed in order to drive the extra load. As for wrapping, we are referring to inserting an instrument that not only can observe the state of a particular signal or port; it will also introduce an extra multiplexer to have the ability to drive the signal to a particular state. This is illustrated in Figure 4.2. The r1500, rWrap and rMatrix are wrapping logic and CMUX is used for tapping.

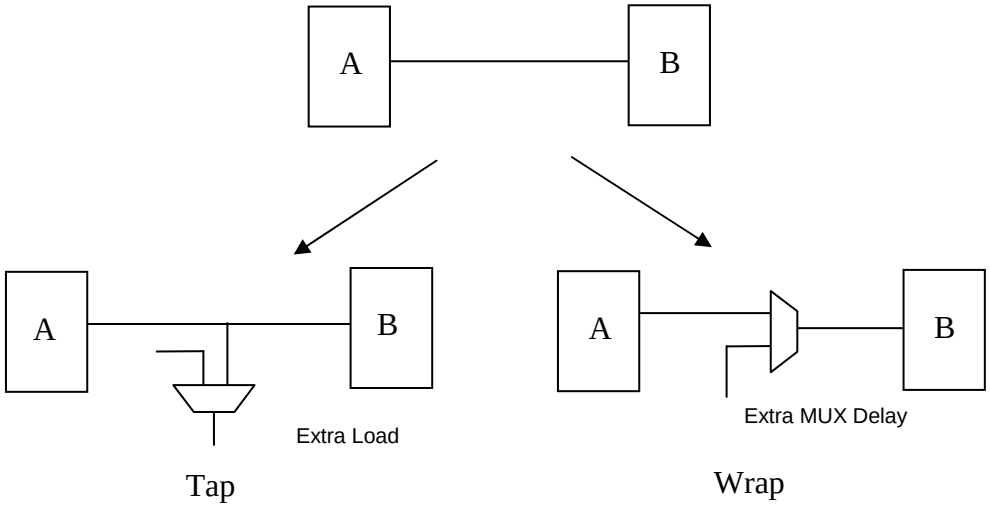


Figure 4.2 Tapping Vs Wrapping Signals

4.5 Instrumentation Considerations

As we discussed in the previous section, the instrumentation of the ReDI block will result in a larger design size. The number of extra gates depends on the choice of ReDI blocks and instrumentation decision. Designers should have a basic guideline in the design process to decide what signals or ports that needs to be instrumented and what kind of wrappers should be used.

As a principle consideration, it makes perfect sense that the debug logic should be inserted in relative new and more complex logic, which has a high risk of failure and may be harder to be verified. In addition, more complex blocks might need more complex debug logic such as the rMATRIX. In our system design, since we have pre-verified the IP blocks in the IP library, they are less likely to have problems after silicon manufacture. Instead, the critical control logic (e.g. AMBA interface, APB Bridge, AHB arbiter) needs to be wrapped since they are more problem prone. Especially we should give more consideration to the control

signals because logic errors are more likely to happen in a design of state machines, while data transfer are more straight forward and do not have much room for an logic error to occur. As a result, our plan is to wrap most of the state machines in the design while tapping the AMBA data bus to monitor the data transfer. Since we would like to fabricate the design with as much as possible debug logic in it, but at the same time the silicon space is limited, so we tried to minimize the size of the on-chip processor and the cache configuration.

Another consideration is that although we would like to test the DAFCA debug approach, we could not predict whether the error will occur in silicon or not. In a case that no design or fabrication error occurs after silicon, we still need a mechanism to test the ability of DAFCA debug method. To solve this problem, we decided to introduce an extra rMatrix attached to the AMBA bus serving as a second IP block. Although this is not what rMatrix was designed for, it can serve the purpose of an embedded FPGA-like block. This gave us the ability to load different designs into the second IP block after the chip is fabricated. The advantage of it is that we could load a design with intentional errors to emulate a design error. Furthermore, a reconfigurable IP block can also perform a different function in the system to enhance the flexibility of the design.

4.6 Instruments Example

Figure 4.3 shows an example of DAFCA instrumentation. Five wrappers have been inserted into the design to wrap the input/output of different cores. Each wrapper consists of a number of reconfigurable logic blocks connected to each other to form a “wrapper” around the core. A SAN is inserted for each wrapper to provide access to the wrapper for other DAFCA on-chip instrumentation. All the SANs in the chip are chained in a serial fashion and connected with the primary controller to form the serial access channel. The primary controller is also connected to on-chip JTAG interface in order to communicate with off-chip environment.

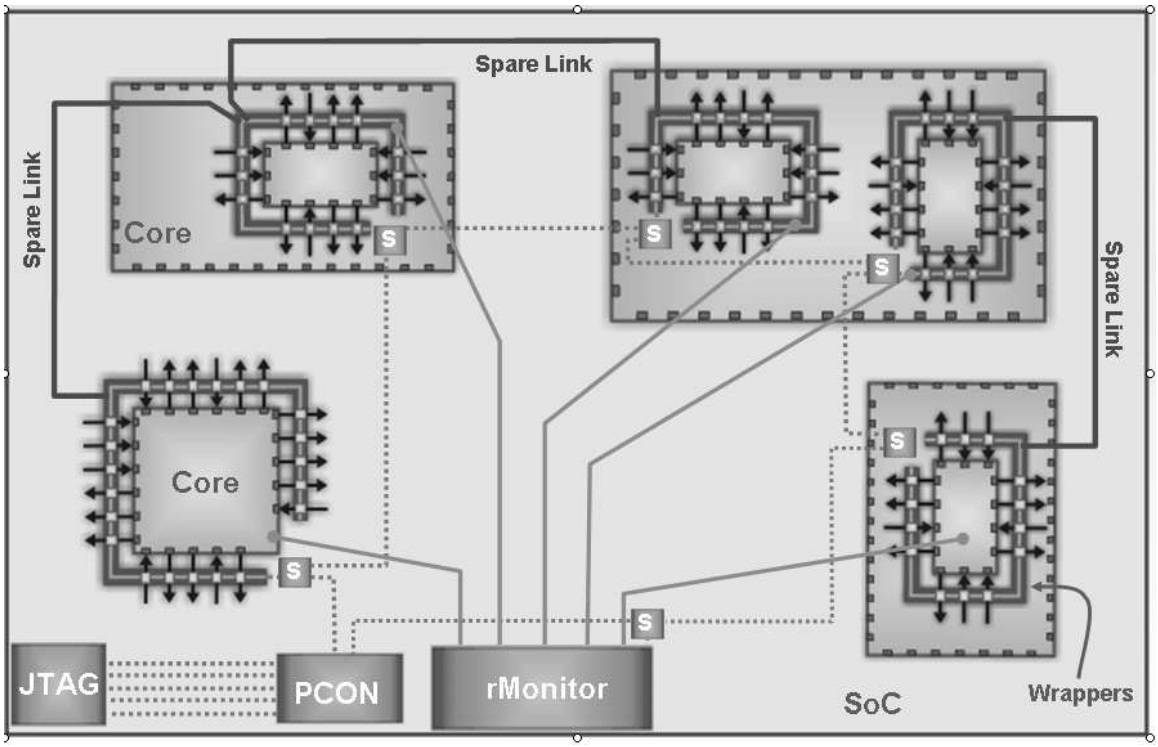


Figure 4.3 Instrumentation Example

An rMonitor is also inserted in the design. It also has a SAN attached and the programming of the rMonitor is also done through the serial access channel. As we mentioned before, the rMonitor can work as an on-chip logic analyzer. One can setup traps and triggers for different wrappers to perform a particular function when it's triggered. The connections between wrappers and the rMonitor can be achieved by directly connect different wrappers to the rMonitor in a star topology or they can be chained together as a daisy chain with multiplexers to avoid routing congestion.

Another thing we should notice from Figure 4.3 is that one can also connect different wrappers with spare links in the design. A big advantage of these connections is that since the wrapper is programmable, it has the potential to fix broken links between core A and B or even perform more complicated logic to fix some errors.

4.7 Post Silicon Tools

In a post-silicon environment, the ‘personal editor’ package of the DAFCA tools will give the designer the ability to modify the behavior of different instruments. The user can program the traps and triggers for the wrapper to execute a particular function, or program the wrapper to bypass all the instruments as in “mission mode”. The post-silicon tools will generate the bit stream and configure the ReDI blocks through the JTAG interface. After the designer has located the error, if there are enough resources inside the chip, one can program the instrument blocks to fix an error. The internal state of the circuit can be recorded in the tracer memory and later be viewed in a waveform viewer. The whole debug procedure can be run in-silicon at-speed.

Chapter 5 Implementation

The entire implementation of our SoC design consists of several major parts:

- Building the library of IP blocks to our specification
- Customizing and verifying the functionality of Leon-2 processor
- Integrate and verify System-on-Chip baseline platform
- DAFCA Reconfigurable IP Instrumentation and functionality verification
- Logic synthesis, Timing Analysis, Physical place and route to final design tape out

In this section the detailed implementation steps of the design are described. At each step of the implementation, verification is done with simulation to ensure the correctness of the final design. The design is implemented targeting a CMOS 180 nm process. The target process can be changed easily since the design is technology-independent.

5.1 Environment Setup

The implementation includes simulation, logic synthesis, physical place and route, RAM library generation, ReDI blocks instrumentation, and post-silicon environment simulation. In addition, both VHDL and Verilog are used in the design and DAFCA instrumentation also uses C++ function calls during post-silicon simulation. Lots of design steps are taken and several CAD tools are involved. Some considerations are also given to integrate different tools for the whole design process.

For logic synthesis, we are using Design Compiler from Synopsys, because it is still the most popular and almost standard logic synthesis tool used in industry. In the physical design step, we are using Cadence SoC Encounter, which is also one of most popular physical design tools for rapid prototyping. For simulation we are using both ModelSim from Mentor Graphics and NCSim from Cadence.

Because DAFCA post-silicon simulation uses several C++ function calls, we chose NCSim for its Programming Language Interface (PLI) support. DAFCA instrumentation was done with DAFCA pre-silicon tools at the register transfer level. After the design was instrumented, the DAFCA post-silicon personality editor was used to program the on-chip reconfigurable logic.

5.2 Building the IP Library

The IP library was built by the entire class of ECE 651 in 2003 and 2004. Students were divided into teams to build up a library of IP cores that can be integrated into our SoC, including FFT, FIR, and AES etc. Each block has been through the whole design flow with pre-layout simulation, logic synthesis, place/route and post-layout simulation. The interface standard and guidelines were established in order to enable the easy integration of the library IP blocks into our system. Each core needs an AMBA interface wrapper to act as a master on the AHB and slave on the APB. With the standardization of the IP interface and general communication specifications, the AMBA interface can be reused with very small modification. The guideline is listed below in detail:

- 32 bit address width
- 32 bit data width
- Reset signal to initialize internal registers and RAMs
- Go signal for IP blocks to start functioning
- Done signal to indicate output data is ready

5.3 Customizing Leon2 Processor

5.3.1 Configuration

Leon2 can be customized for a certain application or target technology. A graphic configuration interface based on Linux kernel tkconfig scripts is provided with the Leon2 Package. After a configuration is saved, the corresponding VHDL file (device.vhd) will be modified and installed. A minimal configuration was chosen.

Optional components such as FPU, Hardware Multiplier, and Debug Units were removed from the configuration. Instruction and data cache were set to be 1K, single set and 8 words per cache line. Targeting technology was also chosen at this point to be TSMC18.

5.3.2 Artisan RAM

The instruction/data caches are implemented as Single Port RAM (SPRAM) blocks and the register file is designed as two Dual-Port RAM (DPRAM) blocks. They are technology-dependent and have to be instantiated by the user. The RAM blocks that come with the LEON2 package are behavior models that can be used only for simulation. In order to synthesize the design we have replaced the RAM blocks with the ones from the Artisan TSMC18 process. The size of the RAM block needed depends on the size of I/D cache and register file. Two 136x32 DPRAMs, two 256x32 SRAMs and two 32x30 RAMs are needed as hard macros in our design.

The Artisan RAM generator was used to generate the RAM blocks. Five views were generated: Verilog Model for simulation, Synopsys Model for logic synthesis, TLF Model for timing analysis, VCLEF footprint for Physical Design tools and GDSII layout for final tape out. The specifications used are shown in Table 5.1.

As we discussed in section 3.6, the RAM blocks cannot be used directly because of the timing specifications difference between Artisan RAM and Leon. We developed a RAM block wrapper to wrap them so that they can be communicating with LEON2 in the same fashion that the behavior model does. Then the Synopsys model is converted into Synopsys database format to be used for logic synthesis.

Table 5.1 Block RAM Parameters

Components	Register File	Cache Data	Cache Tag
Instance Name	dpram136x32	ram256x32	ram32x30
Depth	136	256	32
Width	32	32	30
Frequency (MHz)	50	50	50
Multiplexer Width	4	4	8

5.3.3 Simulation and Synthesis

The Leon2 package comes with a basic testbench that we can simulate. The testbench performs basic tests for the whole system such as memory interface, cache, register file and peripherals. Also, there is a synthesis script that can be used with Synopsys Design Compiler. Pre-synthesis and post-synthesis simulations are done with an Artisan RAM module to make sure the wrapper is designed correctly.

5.4 System-on-Chip Integration

The next step is to integrate user IP blocks to form a baseline system. This is done by designing an AMBA interface and modifying necessary files for Leon to recognize a new bus master/slave. For our baseline design, we are using a 128-bit AES decryption module as user IP. As we mentioned in the previous section, the user IP blocks are defined as the additional master on the AHB bus and slave on the APB bus. We instantiated the user IP block in the top level VHDL file (mcore.vhd), in order for Leon to recognize the extra master on AHB. For the APB connection, we modified the APB/AHB Bridge to define the memory mapping for the APB signals of AES block.

5.4.1 AMBA Interface

The AMBA interface is designed to take care of AMBA signals and pass the input/output data between the AMBA bus and the AES module. It is a state machine to perform several sequential steps for a decrypting call:

1) Initialize, Enable AES module and wait for start request signal on APB
2) Read in control data through APB (input data memory address)
3) Request AHB bus. Transfer data upon bus granted
4) Load key/data into AES and signal AES to start
5) Read in AES output when AES finishes, signal system that output ready
6) Write back output data to the memory address that system specified.
7) Go back to initial state and wait for next call.

5.4.2 Incorporating Bus Masters/Slaves

In order to have the system recognize the additional bus masters and slaves, we had to connect the IP blocks to the bus arbiter and modify the APB Bridge to re-define the peripheral memory mapping.

The IP blocks were instantiated and connected to the AHB arbiter with a unique index. The arbiter's arbitration scheme is fixed and a higher index has higher priority. The CPU is the default master on the bus and has an index of zero, which means that the IP block has higher priority than the CPU when both of them are requesting the bus at the same time. To add APB slaves, we had to connect all the IP blocks to the APB Bridge and re-define the memory mapping of the peripherals in the bridge. As we see in Table 5.2, the memory range of the APB Bridge is between 0x80000000 and 0x8FFFFFFF, and a portion of this memory range is assigned to IP blocks and the rest is for other peripherals (Timers, UART, Cache/Memory Controller etc,). The memory mapping and AHB master indexes of two IP blocks are shown in Table 5.2.

Table 5.2 IP Block APB Memory Mapping

IP blocks	Memory Range	IRQ	Index
rMATRIX	0x80000200-0x800002FF	15	2
AES	0x80000300-0x800003FF	13	1

5.4.3 Synthesis/Place & Route/Simulation

The design was synthesized in the same fashion as we did with Leon alone. In order to verify that the integration of the IP block was done correctly, simulation was done at both the pre-synthesis and post-synthesis steps. In order to test our design, we have modified the C testbench from LEON and recompiled the RAM image for simulation. After Leon boots up, it reads the data from memory, sends it to the AES block through the AHB bus and enables the GO signal through the APB bus. AES reads in the key and encrypted text, performs the decryption and sends the data back to RAM and flags the DONE signal.

After logic synthesis, this original baseline design netlist was loaded into Cadence SoC Encounter for physical design. In this step, we specified the floorplan, synthesized the clock tree, placed and routed the standard cells with optimization until we obtained the final layout and delay file. The delay file was used to perform back annotation simulation in ModelSim to verify the correctness of the design. The details of the physical design procedure will be discussed in section 5.6. The post-layout simulation is shown in Figure 5.1. It can be seen that at around 168μs, AES received data from APB bus. The apb_write data is the external memory address that stores the encrypted data. At around 171μs, the AES requested the AHB bus. It is granted around 172μs and data are transferred through AHB bus. After the data are read, they are converted into 128-bit encrypted cipher text and 128-bit key and sent to the AES module. At 175μs, the AES module finishes decryption and the plain text is written out as data_out and

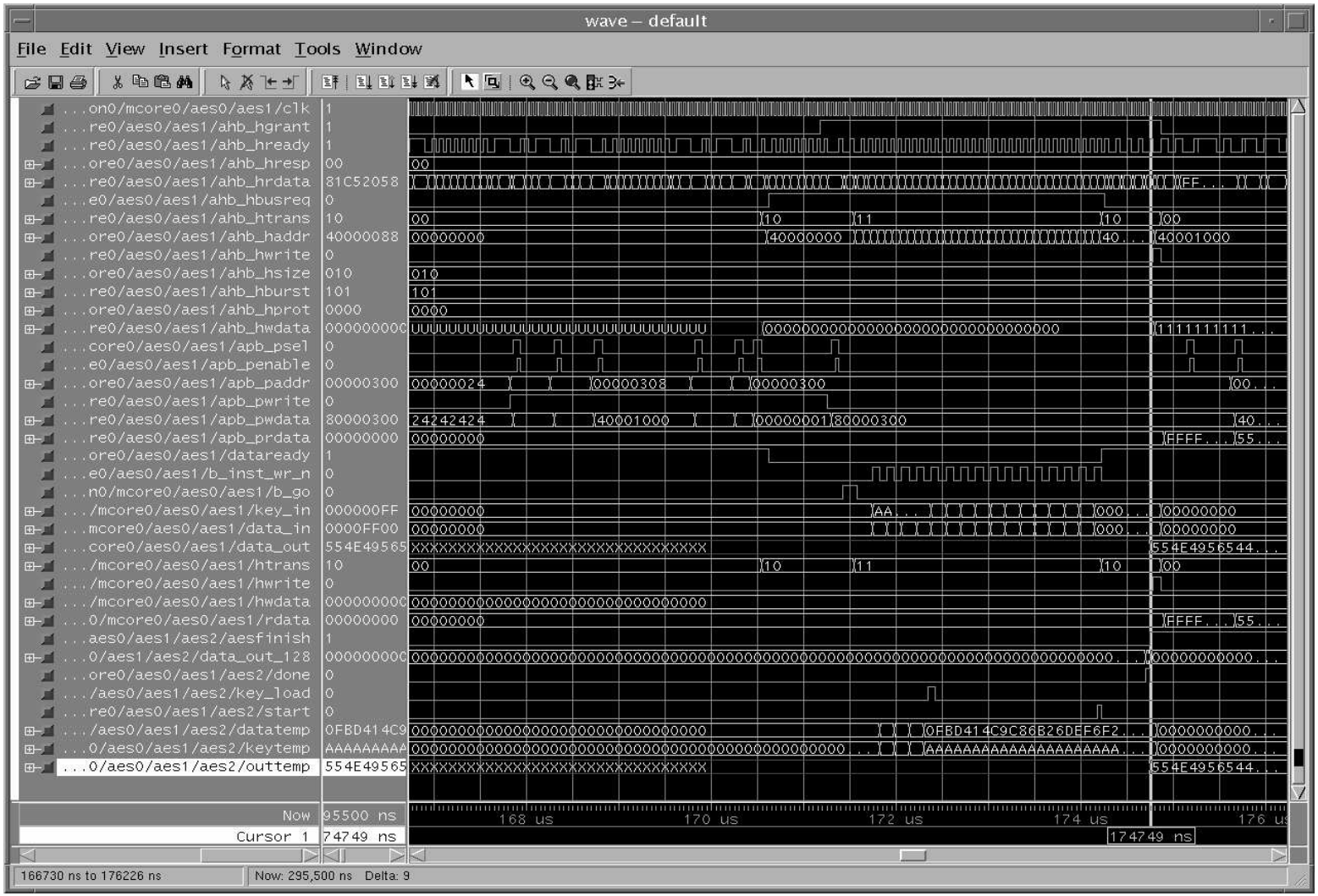


Figure 5.1 Original SoC Post Layout Simulation

aesfinish is high to signal the AMBA interface that the data is decrypted and ready to be sent back to memory.

5.5 DAFCA Instrumentation

The instrumentation should be inserted at the part where we expect to be problematic after the chip is fabricated, so it makes perfect sense to put debugging logic around the most critical part of the design. Since the IP blocks are assumed to be fully tested, we did not instrument anything inside the user IP module. Instead we have wrapped the AMBA bus interface since it's a relative complex state machine. The AHB bus arbiter and APB bridge were wrapped as well. The data bus was tapped to monitor the data transfer. The instrumented design is shown in Figure 5.2. After the design was instrumented, simulation was also done to make sure that the insertion of the DAFCA IP blocks did not affect the logic of the design.

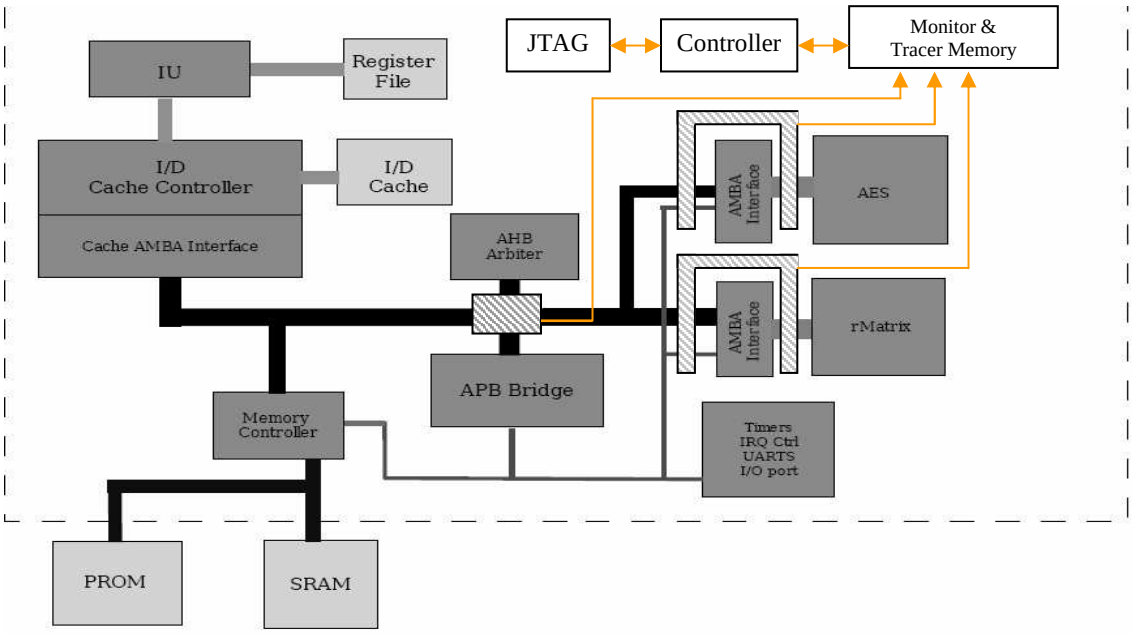


Figure 5.2 Instrumented SoC

5.6 Synthesis

Logic synthesis was done after the DAFCA IP blocks were inserted at the register transfer level. The design was loaded in the Synopsys Design Compiler and constrained with tighter timing constraints. The reason that we are using more constraints is because the addition of DAFCA IP block will introduce extra timing delay. When we perform post-silicon debugging, we might configure the wrapper to perform some extra logic function. If we want to run the circuit at the same speed as the case without the instrumentation, we need to have the circuit designed to run at a higher speed.

Another issue during logic synthesis at this point is that we are using a bottom-up synthesis approach for the on-chip debug module. Because the muxes and D flip-flops are instantiated in the CMUX design and we have a large number of signals being tapped, a large number of muxes and D flip-flops are instantiated in the design. Uniquifying all the sub-modules will lead to extremely long synthesis time and huge amount of memory usage. By using a bottom-up approach we only synthesized each sub-design once and there is only one copy loaded in the memory for each sub-module. After the debug module is synthesized, we can go back to a top-down approach for the rest of design in order to have better optimization. After the design is synthesized, a gate level Verilog netlist is written out with the timing constraint file (SDC format) for timing analysis in physical design.

5.7 Physical Place and Route

After we have the gate level netlist, the next step is to place and route the design to get the layout. The physical design process was done using Cadence SoC Encounter. In this step several key steps were also taken including floorplanning, clock tree synthesis and timing analysis. The details of the design steps are discussed in the following section.

5.7.1 Physical Design Flow

The physical design flow is shown in Figure 5.3. After the synthesized netlist was imported, we started with the floorplanning. In this step, we pre-placed the hard macro blocks (i.e. memory) and floorplan guide as guidance in the placement. Then we proceeded with standard cell placement. After all the cells were placed, we synthesized the clock tree to minimize the clock skew. At this point we used trail route and timing analysis to check if the timing requirement was met. Very often we will have timing violations so several iterations of in place optimization (IPO) are needed. The whole process is an iterative procedure and we might need to go back to modify the initial floorplan, until the timing requirement is met and we can continue to generate the final layout as well as timing delay file for post-layout simulation.

5.7.2 Floorplanning and Placement

After the design was loaded into Encounter, memory blocks (register files, caches) were placed in the corner and aligned. Tracer memory was placed away from the rest because we would like to separate the original baseline design with the extra reconfigurable logic. A power ring was added around the design core area and also around the blocks. Power stripes were used for easy access to VDD and GND. Several floorplans were tried in order to optimize the performance of the circuit.

The basic principle we used in the floorplan was to arrange the modules such that blocks with connections are placed closer to each other, especially for critical components such as CPU, cache controller, memory controller, etc. At the same time, we have tried to separate our original un-instrumented baseline design with DAFCA instruments, to minimize the impact of introducing extra logic into our design.

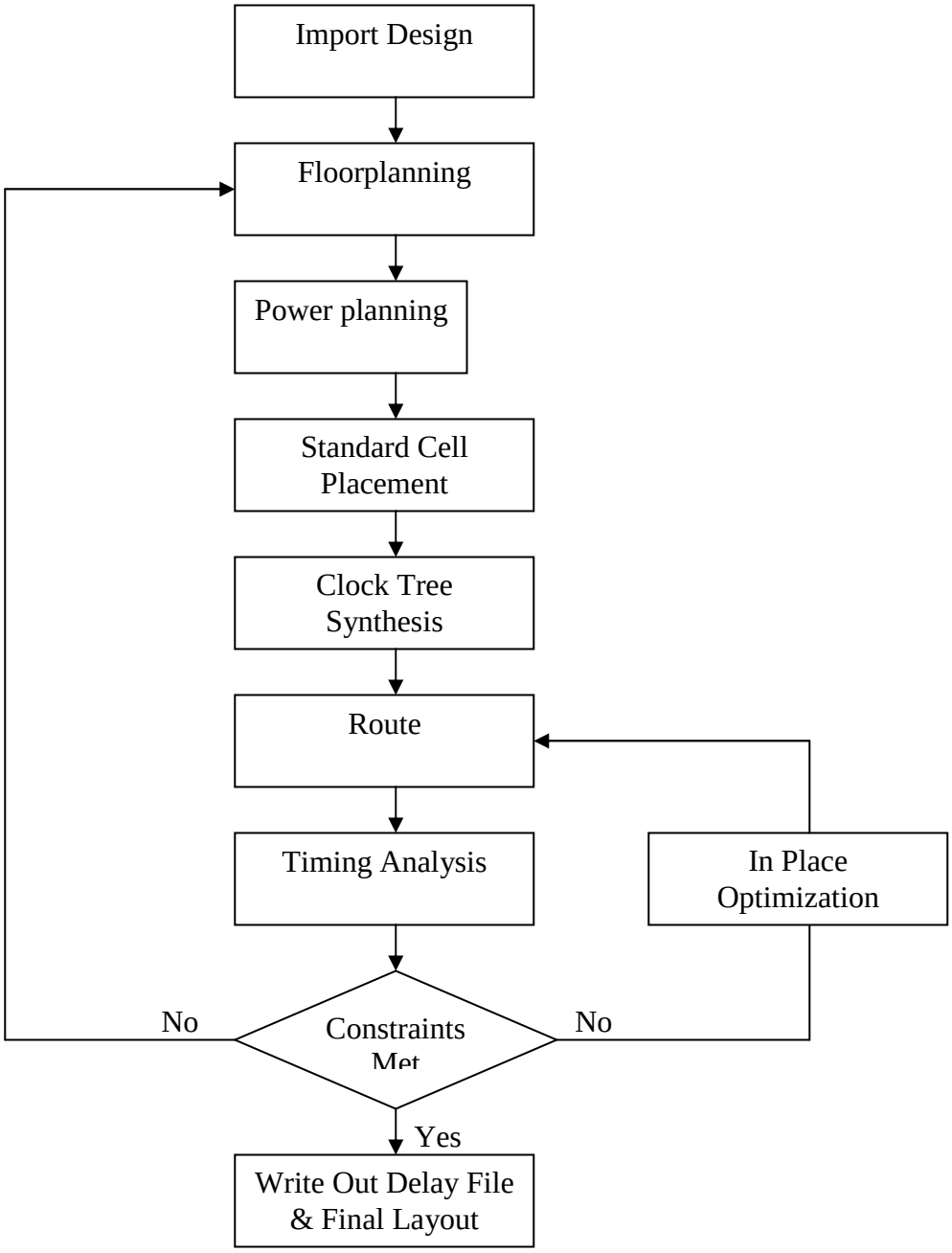


Figure 5.3 Physical Design Flow

Figure 5.4 shows the floorplan guides for our layout. Cache blocks (Blue squares) were placed in the bottom left corner, while register file RAM blocks were placed in the middle left. The CPU module was placed in the left bottom corner since it is using register files and caches. The AES block was placed in the bottom right corner, while the tracer memory and rMonitor were placed in the top right corner to separate the original design with DAFCA instruments. Figure 5.5 shows the final floorplan after the placement. As we compare it with Figure 5.4, we can see that actual placement is consistent with the intention of our floorplan guide. DAFCA blocks and baseline system are mostly separated.

5.7.3 Clock Tree Synthesis

The clock tree was synthesized after the placement is done. The clock tree specification input for Encounter is listed below:

AutoCTSRootPin	*clk*
NoGating	*NO*
MaxDelay	*2ns*
MinDelay	*0.5ps*
MaxSkew	*0.5ns*
SinkMaxTran	*3ns*
BufMaxTran	*3ns*
Buffer	*BUFX1 BUFX2 BUFX3 BUFX4 BUFX8 BUFX12 BUFX16*

The max delay for clock signal was defined as 2ns, with 0.5ns skew. After the clock tree was synthesized, the modified netlist was saved and clock phase delay is shown in Figure 5.6.

5.7.4 Routing

The design was routed using 7 layers of metal. Timing analysis and in placement optimization was done to solve the timing violation after the routing. The final layout of the design is shown in Figure 5.7.

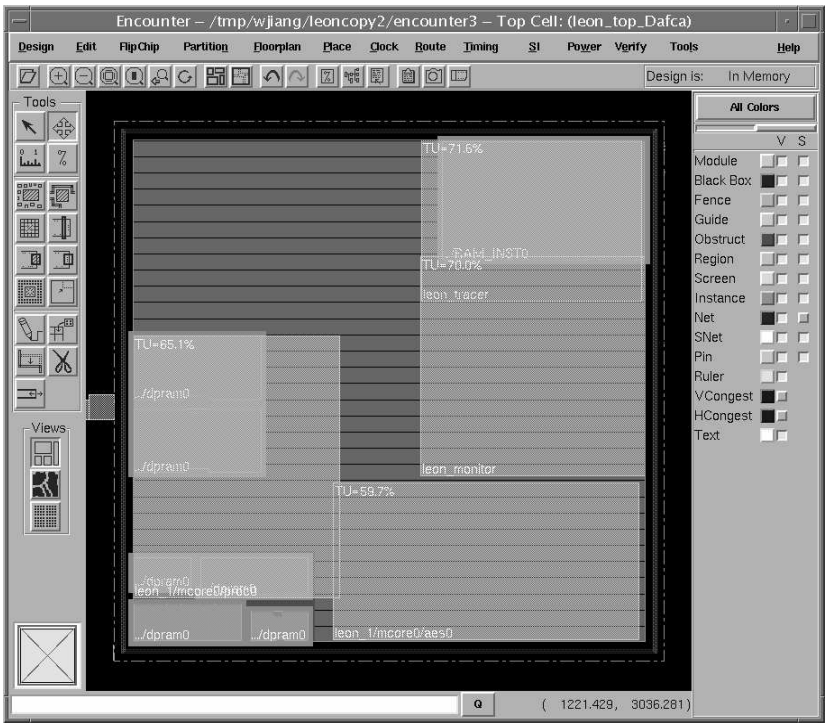


Figure 5.4 Floorplan Guide

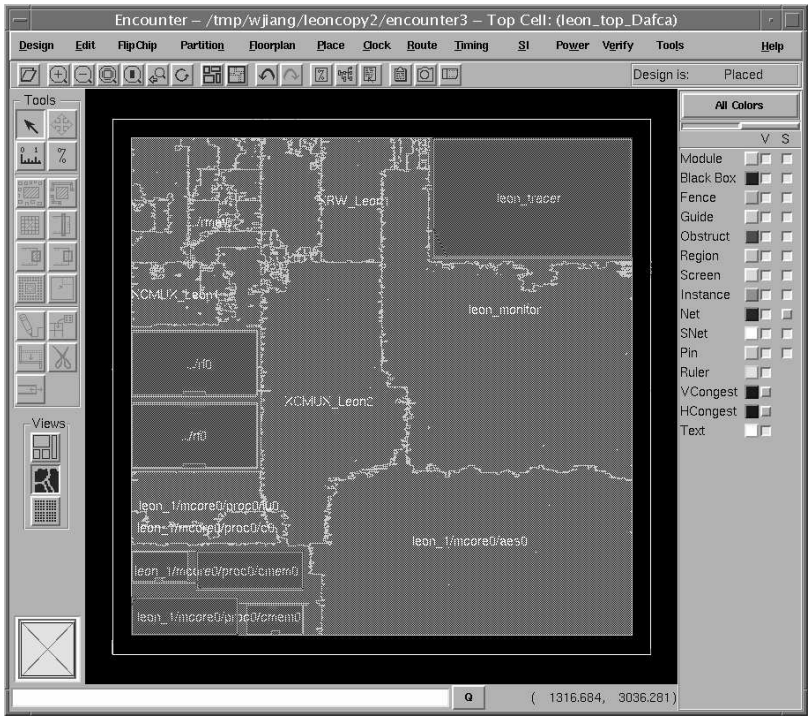


Figure 5.5 Final Floorplan

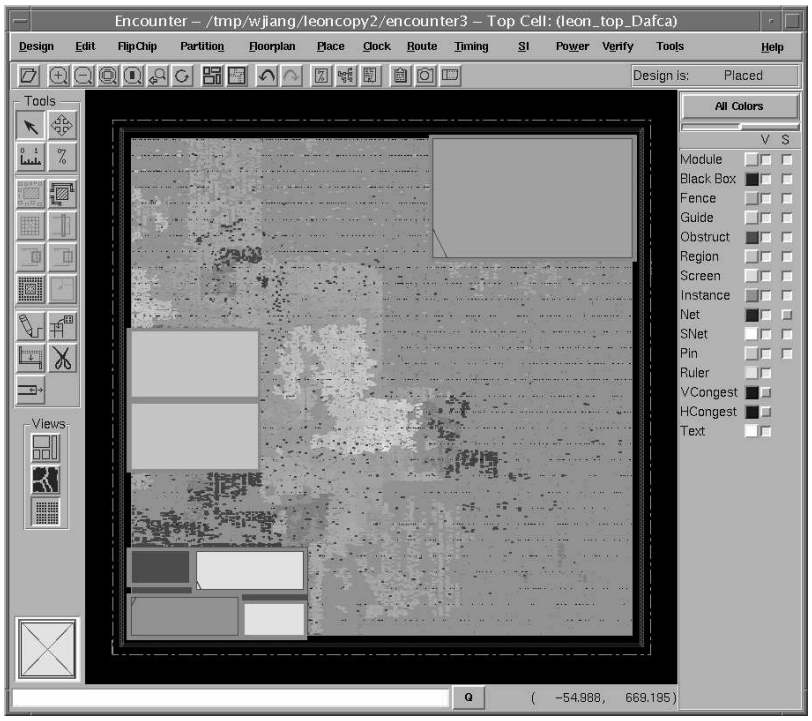


Figure 5.6 Clock Tree Phase Delay

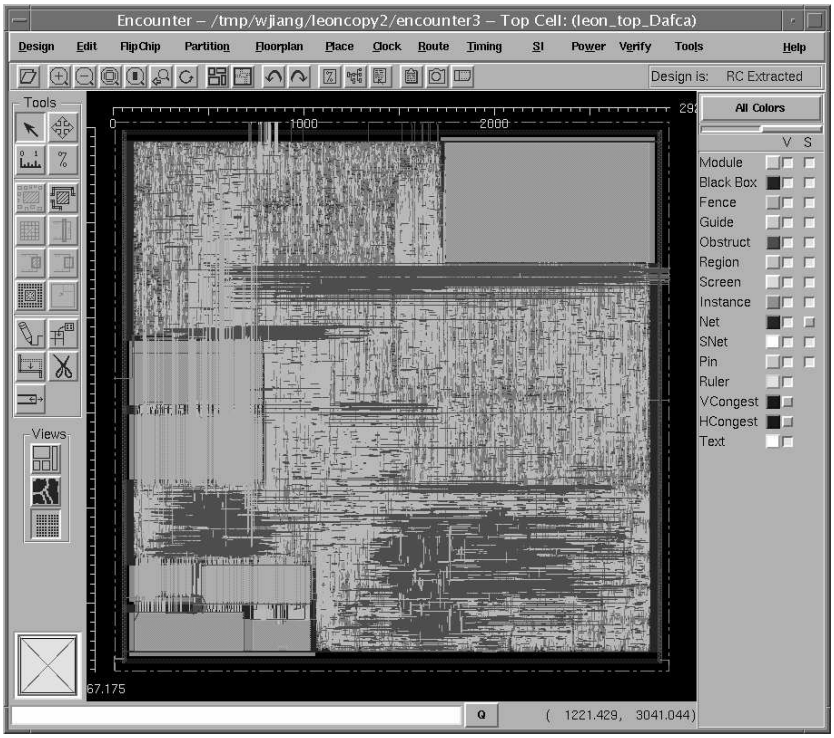


Figure 5.7 SoC Design Routed

5.8 Simulation Result

After the design was routed to the final layout, the timing information was extracted and written to a file in the Standard Delay Format (SDF). The SDF file was imported back to the simulator for back annotation simulation. The simulation waveform is shown in the next figure and from the simulator console we can see that the correct decryption result is printed out. We also performed post-layout simulation with post-silicon tools and the waveform is shown in Figure 5.8.

In our simulation of the post-silicon environment, the design was simulated in a simulator instead of the real chip. Post-silicon tools from DAFCA were used to program the test structure by injecting a personality into the rWrap. Post-silicon tools communicate with the Cadence simulator as if it's a real chip. In our test case, the testbench was modified to loop infinitely. Initially, we chose to bypass the testing structure. The circuit was functioning normally. Then we programmed the testing logic to force the control bits of the AES AMBA interface to be stuck-at-1. As you can see in Figure 5.8, the AES stopped functioning. Then we re-programmed the rWrap to bypass all the testing logic. The AES functionality was restored.

5.9 Discussion

As we can expect, the extra flexibility and debugging ability comes with the price of performance and area. The amount of reconfigurable logic introduced in a SoC design is always about the tradeoff between the flexibility and the area/delay of the design.

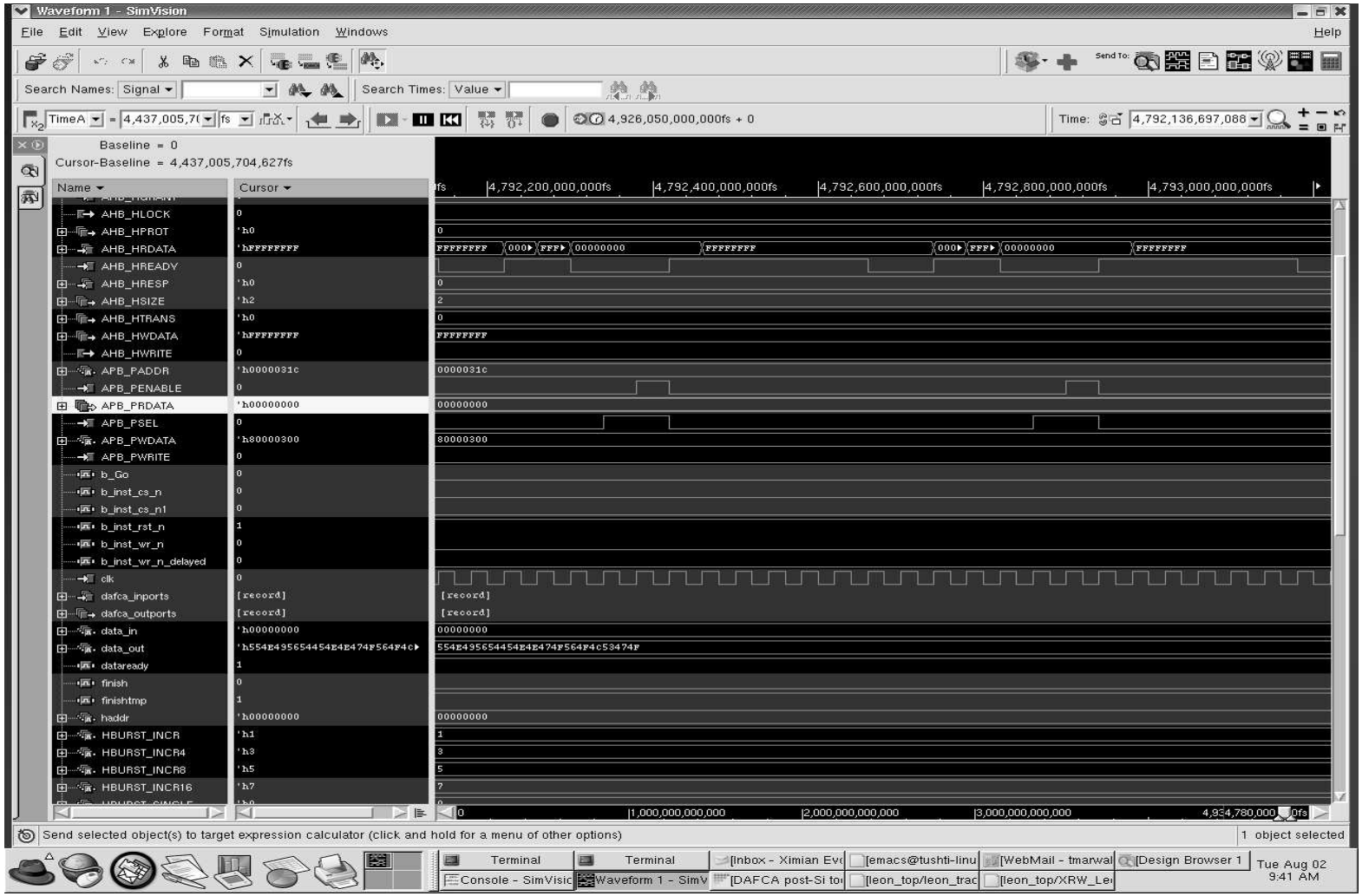


Figure 5.8 Post Silicon Tools Simulation

5.9.1 Area Comparison

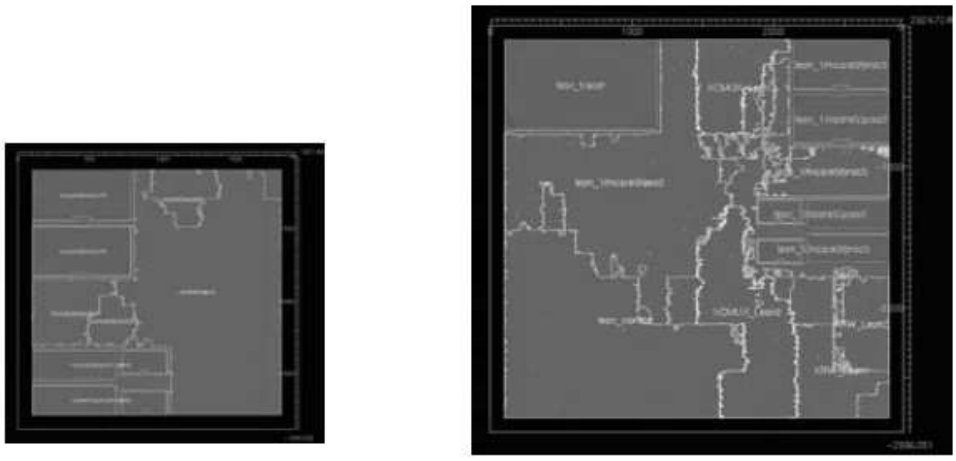
Figure 5.9 is the comparison of the floorplans between the instrumented design and our original design. The un-instrumented design has 812K transistors and the design core fit into a 2mm x 2mm area. For our instrumentation, we have tapped about 400 signals and 100 wrapped signals together with a 2048x44 bit of tracer memory. The total transistor count of the final design is 2048K; the whole design area takes about 3mm x 3mm. The transistor count comparison for different modules is listed in Table 5.3.

5.9.2 Timing Overhead

As we discussed in the previous sections, wrapping a signal will introduce an extra MUX delay. Although tapping a signal does not explicitly introduce any MUX delay in the signal path, the extra load will slow down the transition time and the extra logic will introduce extra wiring delay as well. In the physical layout we have tried to separate our original design and the embedded DAFCA blocks to minimize the impact of the design timing due to the extra wiring. The impact due to the extra load was also reduced since the layout tool performed transistor resizing during timing in-place-optimization. Furthermore, during the synthesis step we used a tighter timing constraint to leave some margin for the extra logic in the ReDI blocks. As in our result, when the circuit is running in mission mode (by-pass DAFCA logic), we did not see any degradation in design performance.

5.9.3 General Discussion

From the result above, we can see that this design approach achieved the primary goal of the project: re-obtain observability and controllability that were lost in SoC design due to large scale integration. It gave the designer the internal access to the chip. We could enable, disable or override internal signals, as well as perform logic fix and record internal state of the circuit. The instrumentation can be integrated into the standard design flow and post silicon software support is already available to utilize the embedded test logic.



Our Baseline SoC Platform
812K Transistors
2 mm x 2 mm

Instrumented Platform
2048K Transistors
3 mm x 3 mm
400 Tapped Signals
100 Wrapped Signals
2048x44 Tracer Memory

Figure 5.9 Area Comparison Between Instrumented/Un-instrumented Design

Table 5.3 Transistor Count Comparison

	Original Baseline	With Test Logic
LEON CPU	408 K	400 K
AES	356 K	328 K
AMBA Bus	48 K	52K
Cache	140 K	140 K
Register File	188 K	188 K
RAM Total	328 K	328 K
wrapper1		64 K
wrapper2		92 K
CMUX		370 K
rMonitor		392 K
Tracer Memory		288 K
SOC Total	812 K	2040k

From cost perspective, as shown in the result above, about 60% of our design is embedded test logic, which might not seem very attractive in this particular case. But it should be noted that the final design has only about 2 million transistors, including 700K transistors that are used for on chip debug module (rMonitor and tracer memory). In a multi-million gate design, the size of the on chip debug module will remain the same while only the size of the wrappers will be scaled. In general, this approach will be more suitable for a large SoC design where the extra cost can be justified.

Another consideration of embedding test logic in a design is that the designers will have to make the decision where to insert the test structure. In the future, a quantitative measurement of the quality of designer's decision will be needed to help the designers in understanding the impact of the instrumentation.

Chapter 6 Future Work and Conclusion

6.1 Future Work

In the design a dummy module of rMATRIX is used as the second IP block to verify the design of the rMATRIX AMBA interface. The insertion of the second IP block (i.e. rMATRIX) can be done by wrapping the input ports of a dummy module with rMATRIX block. The size of rMATRIX was set as 9x2 because of limited space in silicon. The design flow needs to be repeated to obtain the layout with rMATRIX.

More post-layout simulation needs to be done, especially with DAFCA post-silicon tools. More stress needs to be added in the simulation to obtain higher confidence in correctness of the design before its fabrication.

The design is planned to be submitted for fabrication through MOSIS. A testing board needs to be designed and built before the silicon is back. The fabricated chip should go through the same tests as in post-layout simulation. Moreover, with rMATRIX as an IP block in the chip, different designs can be loaded in the chip to test the efficiency of our SoC verification/debug plan.

6.2 Conclusion

- Developed an open baseline SoC Platform that can be used for enhancement or derivative design.
- We have instrumented the platform to embed test reconfigurable logic for in-silicon debug and system verification.
- We have simulated the instrumented design in post-silicon environment and verified the basic functionality of the test logic.
- Platform based approach will help the designers to achieve higher quality design as well as reduce the product design cycle.

- Embedded reconfigurable logic will enhance the design by providing flexibility and reliability. Rapid prototyping can be achieved with in-silicon at-speed debug/repair.
- Timing and area overhead are cost effective in multi-million-transistor SoC designs because re-spins and product delays may be avoided.

References

[1] Moore, G., “Cramming more components onto integrated circuits”, Electronics Magazine, 1965

[2] Intel, “Moore’s Law 40th Anniversary”, [Online]. Available: http://www.intel.com/technology/mooreslaw/

[3] Moore’s Law, [online] Available: http://en.wikipedia.org/wiki/Moore's_law

[4] International Roadmap Committee, ITRS roadmap 2003 Edition, [Online] Available: http://public.itrs.net/Files/2003ITRS/Home2003.htm

[5] Bouldin, D. ECE 551/552/651/652 Class Note. [Online]. Available: http://www-ece.engr.utk.edu/ece/bouldin_courses/index.html

[6] Peterson, G. ECE 659 Digital Systems Verification Class Note, Spring 2005

[7] Miczo, Alexander, “Digital Logic Testing and Simulation”, John Wiley & Sons, Inc. 2003

[8] Bouldin, D. “Platform-Based System-on-Chip Design”, Proceedings of 2003 Microelectronic System Education Conference (MSE), Anaheim, CA, pp. 48-49, June, 2003

[9] DAFCA Inc., “In-Silicon Solution for Silicon Debug”, DAFCA whitepaper.

[10] Srivastava, R., ”Development of An Open Core System-on-Chip Platform”, M.S. Thesis, University of Tennessee, August 2004.

[11] DAFCA Inc., “More Bugs in Silicon and How Will You Deal With Them”, Whitepaper

[12] DAFCA Inc., “DAFCA ReDI Instruments Data book”

[13] Xilinx Chipscope™ Manual

[14] Synplicity Identify™ Manual

[15] FS2 Bus Navigator™ Manual

[16] Opencore, [Online] Available: http://www.opencores.org

[17] Jiri Gaisler, Gaisler Research. The LEON-2 Processor: User’s Manual. [Online]. Available: http://www.gaisler.com/.

[18] Rudolf Usselmann, OpenCores SoC Bus Review, [online]. Available: http://www.opencores.org/projects.cgi/web/wishbone/soc_bus_comparison.pdf

[19] Kyeong Keol Ryu; Eung Shin; Mooney, V.J., “A comparison of five different multiprocessor SoC bus architectures”, Digital Systems, Design, 2001. Proceedings. Euromicro Symposium on, 4-6 Sept. 2001, p202 – 209

[20] AMBA Specifications Rev 2.0, ARM IHI 0011A, ARM Limited, (1999). [Online]. Available: http://www.arm.com,

[21] Gaisler Research, LEON/ERC32 Cross Compilation System, [Online]. Available: http://www.gaisler.com/products/leccs/leccs.html

[22] RTEMS Steering Committee, Real-Time Operating System for Multiprocessor Systems. [Online]. Available: http://www.rtems.org/

Vita

Wei Jiang was born in Beijing, China on May 17th, 1977. He received his Bachelor of Science in Electronics from Peking University in 1998. He attended The University of Tennessee, Knoxville in 1999 and received his Master of Science degree in Physics in August 2002. He then joined the Department of Electrical and Computer Engineering at the University of Tennessee, Knoxville and conducted his research under the guidance of Dr. Don Bouldin. He received his Master of Science degree in Electrical Engineering in December 2005. His research interests include VLSI design, digital systems and semiconductor devices.